Presenting Business Insights on Advanced Pricing Agreements Using a Business Intelligence Framework

Tiago Daniel Gradert Marques

Proposal for the Project Report presented as partial requirement for obtaining the Master's degree in Information Management with a Specialization in Knowledge Management and Business Intelligence

NOVA Information Management School
Instituto Superior de Estatística e Gestão de Informação
Universidade Nova de Lisboa

NOVA Information Management School
Instituto Superior de Estatística e Gestão de Informação
Universidade Nova de Lisboa

PRESENTING BUSINESS INSIGHTS ON ADVANCED PRICING AGREEMENTS USING A BUSINESS INTELLIGENCE FRAMEWORK

by

Tiago Daniel Gradert Marques

Proposal for the Project Report presented as partial requirement for obtaining the Master's degree in Information Management with a specialization in Knowledge Management and Business Intelligence

Advisor: Professor Roberto Henriques, PhD.

May 2020

ABSTRACT

In companies that use advanced pricing agreements, pricing managers are responsible for setting the new and adjusted discounts from time to time. These companies are usually of great dimension and so the number of products and customers is extensive, which causes the decision-making to be challenging for the pricing managers. To aid in this process, this project report incorporates a business intelligence framework to model the data into a dimensional model that will provide the pricing managers with business insights by allowing them to have a more targeted and detailed view of the data through multiple contextual perspectives.

The data sources used were provided by a client at BI4ALL and consist of two different JDE extracts: an export of the advanced pricing agreements that include all the pricing rules and an export of the sales data following those pricing rules. Both sources of data will be used to implement a business intelligence framework. The final outcome of this project report is presented in a dashboard with multiple visualizations, where the pricing manager can navigate and obtain data in a dynamic way according to the information requested. This will allow for a better analysis, and thus, for better pricing adjustment and optimization.

KEYWORDS

Business Intelligence; Data Warehousing; Advanced Pricing Agreements; Pricing Rules; Discount Dashboards; Pricing Analysis

INDEX

LIST OF FIGURES

LIST OF TABLES

List of Acronyms

3NF	Third Normal Form
ADPR	Advanced Pricing
APA	Advanced Pricing Agreements
BI	Business Intelligence
CIF	Corporate Information Factory
COGS	Cost of Goods Sold
CTE	Common Table Expression
CY	Current Year
CYTD	Current Year to Date
DBMS	Database Management Systems
DDL	Data Definition Language
DWH	Data Warehouse
EDW	Enterprise Data Warehouse
ERD	Entity Relationship Diagram
MDX	Multidimensional Expressions
OLAP	Online Analytical Processing
OLTP	Online Transaction Processing
PLS	Partial Least Squares Regression
PY	Previous Year
PYTD	Previous Year to Date
RSR	Retail Systems Research
SA	Staging Area
SLA	Service Level Agreement
SQL	Structured Query Language
SSMS	SQL Server Management Studio
SSIS	SQL Server Integration Services
SSAS	SQL Server Analysis Services

1. INTRODUCTION

This report arose from a business need requested by a pricing manager from a client at BI4ALL. The business need consisted in having visibility of the multiple layers of discounts applied on the final price of each invoiced sale that resulted from an advanced pricing agreement. Such detail was not directly available for the pricing managers because the discounts and invoiced sales came from different sources.

On advanced pricing models, agreements on prices between two parties are made in advance in which discounts are defined through pricing rules and, if met, are cast upon the base price of a product. This allows a company to maximize profits as it enables the automation of the execution of various layers of discounts and enforce compliance (Oracle Corporation, 2015). These rules can be applied in various ways and cycle through multiple layers before establishing the final price.

A pricing versus sales comparison can be applied to advanced pricing agreements to evaluate the implications of the discount variations. Pricing implications can affect many outcomes of the business within a company such as sales volume and profits (Lipovetsky, 2011). In this way, an adjustment of a discount rule or even the non-adjustment can lead to a change in consumer behavior (Isabella, Pozzani, Chen, & Perfi Gomes, 2012), for example, to a decrease in sales, an increase in the number of sales but a decrease in net invoice sales, a decrease in customer satisfaction or even loss of clientele. Therefore, a careful analysis should be implemented to the adjustment of the current prices with the new prices of the advanced pricing agreements in order to avoid the negative and improve the positive outcomes. To accomplish this analysis, the data must be prepared and organized in a specific manner and then presented to the pricing manager.

The data presentation is important to enable the pricing managers to get a better understanding of trends, best practices, how different customers react to different price rules and other possible scenarios. This is the objective of this project report, to apply a business intelligence framework on the pricing and transactional data sources in order to present visualizations that will provide new business insights to the pricing managers and promote a more informed and improved decision-making when adjusting the pricing rules.

1.1. BACKGROUND AND PROBLEM IDENTIFICATION

There are two data sources provided by the client, which are JDE extracts of tables in a relational database. One extract contains transactional data and the other advanced pricing agreements data.

The transactional data only provides the list price and final price of the transactions with no detail on which discounts were applied, therefore the challenge is to get that information from the advanced pricing extract. The problem is that the advanced pricing data is presented in form of a list of the valid discounts and rules for each customer or customer group and product or product group in different date ranges, and therefore, the pricing manager had to provide the business logic behind the process on how the final price is calculated. This process needs to be replicated in this project's ETL since it is calculated in the advanced pricing software and is not visible in the extractions.

The business logic behind the advanced pricing agreements extract is the following:

- An advanced pricing agreement is always applied to both a customer or customer group and a product or product group, meaning that no discount is valid solely for a customer or a product.
- There are two types of discounts: special prices (absolute value reduction of the list price) and discounts in percentage.
- A customer and product can have multiple discounts applied at the same time, in which case if both a special price and a discount percentage are applied within the same time frame, the special price takes priority.
- There can only be up to 5 discounts applied at the same time for a customer and product.
- The discounts present on the advanced pricing agreements extract have had their conditions met and are final, meaning that if a transaction occurred on a particular day, all the discounts that are valid on that period will be applied for that transaction.
- When a customer code is blank, the customer group must exist and the discount will be applied to all the customer codes of that particular customer group. Same logic applies to the product codes and product groups.
- Whenever a Customer Code and Product Code are both blank for an advanced pricing agreement, that particular agreement will be applied to all the customer codes and product codes of their respective groups. E.g. if a customer code and product code are blank and their groups are comprised of 12 customer codes and 5 product codes, then that pricing agreement will be applied to 12 x 5 = 60 combinations of customer codes and product codes.

More details regarding the challenge of the business need can be found in subchapter 3.1, where a detailed view of the data sources is presented.

In addition to the main challenge, the transactions extract is granular at the day level while each record of advanced pricing agreements data is comprised of a date range. Even though both will have their own dimensional model, they need to be combined in order to achieve the final result.

1.2. Study Objectives

The study objectives are bound to the ETL transformations responsible for modeling the advanced pricing data. The main challenge in the construction of this dimensional model is to find a way to correctly separate and aggregate the different discounts that are valid in the same instance in time into common date ranges taking into account also the different list prices, which also vary from time to time. The achievement of this objective requires the development of a new mechanism, which must be fully studied and tested. Once this study objective has been completed, its usage can be applied and extended to other similar date range problems. An in-depth analysis of this topic can be found in the dedicated sub-chapter 3.3.2.2 and additional details in subchapters 7.3 and 7.4.

Finally, after both transactional and advanced pricing dimensional models have been created, the final study objective is to combine both models and build the advanced pricing dashboards to empower the pricing managers with the necessary tools and visualizations to be able to do the appropriate analysis and fulfill the business need.

1.3. Study Relevance and Importance

Applying Business Intelligence to the advanced pricing agreements can grant relevant and important information to pricing analysis by providing information that the pricing managers could otherwise not obtain directly or even at all. With the support of business intelligence tools, we can model the data in a dimensional structure which allows us to look at the data through different dimensions and make broad comparisons. This way, the pricing manager will be supplied with more relevant information to make decisions regarding the setting of new discount prices for each customer, making the task easier, targeted and more efficient.

The main challenge around the advanced pricing agreements data is that a new methodology must be implemented and tested in order to correctly segment and structure the data to be able to understand that on a particular day which discounts were applied that took part in the calculation of the final price.

There is a growing correlation between Big Data, Business Intelligence and Business Analytics as shown by (H. Chen, Chiang, & Storey, 2012). With any large volume of data, such as in advanced pricing, there is the need for the business managers and pricing managers to have the appropriate analytical skills and find relevant information in large volumes of data to have the knowledge to make good business decisions. However, (Paris & Washington, 2018) report states that technological innovation has brought a shift in skills needed in the workplace, meaning that many workers are under-skilled and unprepared to work efficiently. The report also informs that there are shortages of specialized information technology workers in several countries in Europe and in the United States. For example, by 2020, France is expected to be short of 80.000 workers in this area of expertise. Furthermore, data analytics is one of the top 3 areas with the largest skill shortage as of 2018.

Arming managers with targeted business insights is critical for a company to innovate and keep up with Big Data, as well as have better chances of succeeding and maximizing profits.

2. THEORETICAL FRAMEWORK

This Project report is focused on two main subjects: first, the understanding of the business need associated with the advanced pricing agreements given the data sources and business logic that are applied; and second, the extract, transform and load processes to model the two data sources into a compatible and organized dimensional data warehouse in order to meet the requirements and deliver the requested business insights to the pricing managers. As such, the theoretical framework will be centralized in the importance and understanding of Advanced Pricing Agreements and in the Business Intelligence state of the art.

2.1. ORACLE ADVANCED PRICING AGREEMENTS AND PRICING STRATEGIES

It's not surprising that price promotions have a major role in an organization's profits and market share. In fact, they play a large and important role in both online and offline marketing campaigns. Competition has matured to the point that in order to maximize profits, promotions must be optimized in a way that individual and customized promotions are made taking into consideration historical sales data (Grewal et al., 2011). For this reason, it's important to get the right promotion of the right product to the right customer.

According to (Grewal et al., 2011), in order to better understand promotion innovations, we must first start by asking whom to target with promotions. One of the key factors that helps answer this question is looking at purchase history data, which retailers use in order to create customizable promotions for customers and customer segments. The second question concerns what price and promotion models to use. There are many new promotion models emerging. Some examples are conditional promotions, where promotions are bound to certain conditions and will apply if certain consumer purchasing activities are met. In this type of promotion, the consumer partly holds the control on whether the promotion will be applied. Other innovative promotions involve volume-based pricing, where larger discounts are applied if a prearranged number of consumers agree to purchase a product within a specific timeframe. Dynamic pricing models are gaining higher usage and giving birth to dynamic pricing software, enabling companies that sell large volumes of products to get a better analysis on price elasticity.

Dynamic pricing is a pricing strategy that changes according to a set of variables that are non-customer related, including time of the day, available supply and can even incorporate the competitors' prices and availability, according to (Baird, 2017). Personalized pricing, however is customer-oriented. It's a unique offer specifically directed at a particular customer. This type of pricing strategy takes into consideration customer loyalty and historical data as opposed to dynamic pricing. Personalized pricing may land the appropriate discount to the customer at the right time that could result in a purchase that might otherwise have been lost. As stated by (Baird, 2017), the RSR's 2017 benchmark on retailers' pricing strategies has shown that there has been an increase in retailer's viewpoint of personalized pricing as an opportunity from 31% 2016 to 33% in 2017. On the other hand, dynamic pricing decreased from 28% in 2016 to 22% in 2017.

Price discrimination can be summarized as charging different customers with a different price for the same product, according to (Elegido, 2012). But there are some conditions that must be met for this pricing strategy to work. First, the seller must have indicators on what is the customer's price reservation, meaning the maximum amount that the customer is willing to pay for the product. Second, the competition of the supply for that product must be low, otherwise the price would be battled over towards the marginal cost, making price discrimination practically impossible. And Lastly, the final price for the first buyer must not be low enough that they can resell it at a higher price, entering in competition with the firm they first bought it from to resell it to their customers with a higher reservation price.

Other pricing strategies include orders in which the customer is eligible for more than one discount for the same product, stacking them together for a lower final price. Examples include using multiple coupons that are allowed to be combined in retail purchases and big corporate agreements that stack discounts by achieving agreed volume purchases.

The third and last question to enhance promotion innovations according to (Grewal et al., 2011) concerns what promotional design elements to use to increase the effectiveness of price promotions. Consumer behavior studies play an important role in designing the promotions. For example, (H. (Allan) Chen & Rao, 2007) have shown that consumers perceive sequential discounts bigger than a single big discount, even though the sequential discounts have the same promotional value as the big one. Consumers are also more inclined to purchase products that have percentage discounts (such as 50% off a product) than absolute discounts (such as \$5 off a \$10 product) for low price products. For large price products, the logic thought is opposite to the consumers as they are more likely to buy a product based on a promotion of \$10 off a \$250 product than a 4% discount on that same product, as demonstrated by (S. F. S. Chen, Monroe, & Lou, 1998) and cited by (Grewal et al., 2011).

Pricing engines are software programs that use a variety of pricing strategies to craft optimal value offering on the portfolio level. They enable price personalization to satisfy customer requests while achieving business goals at the same time. Dynamic pricing engines can incorporate market trends, customer behavior, purchasing power and demand fluctuations, aiming not to replace pricing managers, but to help them make the right decisions (Competera, 2019).

Oracle Advanced Pricing is a pricing engine developed by Oracle Corporation to enable organizations to setup complex and tailored pricing scenarios, allowing for more targeted and optimized prices being sent to customers. It uses a framework that can be broken down into 3 stages: defining rules, assigning those rules to specific actions and defining price controls. The functionalities embedded in this tool are numerous, including creating list prices for products and product hierarchies, calculating prices through dynamic formulas, defining pricing agreements, set usage constraints to limit promotions, and defining pricing qualifiers such as individual customers, customer groups and order amounts (Oracle Corporation, 2010).

2.2. Business Intelligence

Business Intelligence is a term known for characterizing the technologies and processes responsible for gathering and transforming the data in order to help users make better business decisions (Wixom & Watson, 2010). According to (Turban, Sharda, & Aronson, 2011), the primary objective of Business Intelligence is to provide business users with the ability to conduct appropriate analysis on data, providing them with insights to make more informed and better decisions.

With the volume of data increasing each day and the price of computer storage decreasing, companies are storing more detailed and larger amounts of data in multiple storage systems. Business Intelligence technologies allow for massive amounts of data to be organized and structured in ways to facilitate their analysis and understanding. The ultimate goal of Business Intelligence is to enable business users on all levels in a company to make effective and informed decisions. Getting access to all the information is not enough to aid in the decision-making. The user must be able to analyze and filter the appropriate information to attain the necessary business insights and knowledge to make decisions (Ng et al., 2013).

The benefits of Business Intelligence, according to (Turban et al., 2011), involve providing business users with the information to answer their business questions and produce business value. Some of the examples of business value that Business Intelligence analytical applications can provide are: the ability to personalize customer relationships for higher satisfaction and retention; target customers based on their need to increase their loyalty to the product line; increase campaign profitability by focusing on the customers that are most likely to buy a product; make individual interaction decisions based on overall client profitability; and prevent loss of high-value customers and let go of low-level customers.

2.2.1. Data Warehouse Development Approaches

A data warehouse is an organized repository of data to be accessed by end users through applications. It is subject-oriented, combines data from different sources into a standard format, it is time variant because all data warehouses must support a time dimension, and nonvolatile as in end users cannot change or update the data in the data warehouse (Turban et al., 2011, pp. 29-78). According to (W.H. Inmon, Derek Strauss, 2008, pp. 1-22), this definition has been accepted from the beginning, adding that a data warehouse is a collection of data in support of management's decision. They argue that one of the major advantages of a data warehouse is the ability to use the data in the data warehouse to look across an organization. According to (R. Kimball & Caserta, 2004, pp. 29–52), one of the major responsibilities of a data warehouse is to combine data from multiple legacy applications of an enterprise into a single cohesive repository.

There are two worldwide renowned data warehousing experts, Ralph Kimball and William H. Inmon. They offer two different architectural approaches when building a data warehouse. Inmon promotes the hub-and-spoke architecture, while Kimball advocates in favor of the data mart bus architecture with linked dimensions, which will be addressed in chapters 2.2.1.1 and 2.2.1.2.

2.2.1.1. Kimball's Approach

Goals of Data Warehousing

The fundamental goals of data warehousing and business intelligence are derived from the business management needs, as they drive the requirements for the foundations of dimensional modeling, according to (Ralph Kimball & Ross, 2013, pp. 1-35). Having a large volume of data and not being able to access specific parts of the data, not having the important information organized and not knowing who has the right numbers are all questions that serve as requirements that can be answered with a data warehouse and business intelligence system.

The goals of a data warehouse, as stated by (R, Kimball;R, Margy 2008) are bound to:

- Making an organization's information easily accessible. This means that the information in the data warehouse must be fast to query, understand and navigate.
- Making the organization's information consistent, meaning that the information is of high-quality, correct and complete.
- Being an adaptive source of information. An organization is continuously changing and new requests can be made by the business. The data warehouse must be adaptive to change and incorporate those requests without compromising the existing data and what has already been built.
- Protecting the information of an organization. The data warehouse users have access to all kinds of sensitive information that should be controlled.
- Support the foundation for decision-making. The data in the presentation layer aids business users in their decision-making and it comes from the data warehouse.

Dimensional Modeling

According to (Ralph Kimball & Ross, 2013, pp. 1-35), dimensional modeling is the art of making databases simple. With large volumes of data, simplifying and structuring the data for easy access is key in delivering the necessary information to the users. It is extensively accepted as one of the preferred techniques for presenting analytical data because it aims to deliver data that is understandable and also contemplates fast query performance. Dimensional models when instantiated in relational database management systems are referred to as star schemas because their format resembles the shape of a star, and when implemented in multidimensional databases, they are known as online analytical processing (OLAP) cubes. OLAP cubes are superior in delivering query performance because they support indexing strategies, precalculated summary tables and other types of optimizations, but there is an extra load time for the OLAP cubes to be processed, which must be considered. On the front end they allow for drilling down, drilling up and adding or removing attributes with great performance without having to issue new queries. OLAP cubes are often derived from or equivalent in content to relational star schemas. Star schemas are the recommended dimensional model to load atomic data, OLAP cubes are then optionally populated with data from the relational star schemas.

Kimball's Data Warehouse Business Intelligence architecture

There are 4 core elements in the Kimball Data Warehouse Business Intelligence architecture: operational source systems, ETL system, presentation area, and business intelligence applications.

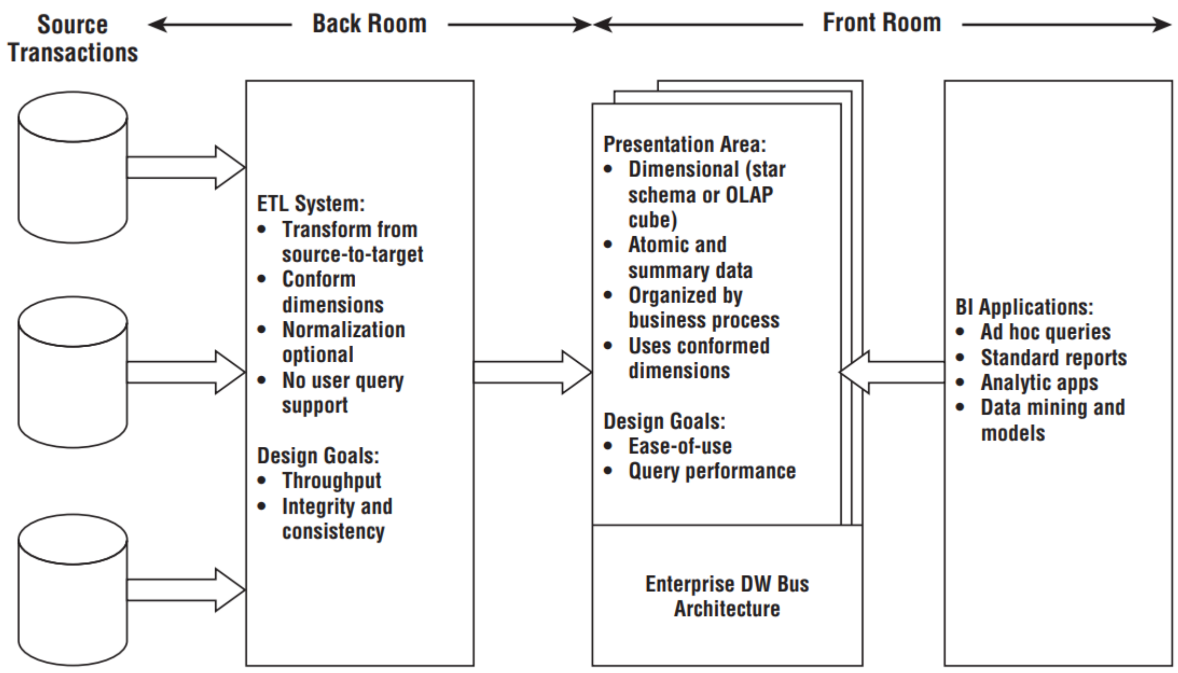


Figure 1 - Core elements of the Kimball DW/BI architecture (Ralph Kimball & Ross, 2013, pp. 1-35)

Operational Source System

Operational Source Systems are systems that record business transactions. It's the data that will be used in the back room of the architecture as input data for the ETL processes. Data sources read data from mainframes, relational databases, flat files, XML sources, Web logs, and enterprise resource planning (ERP) systems, as stated by (R. Kimball & Caserta, 2004, pp. 55-111). These source systems should be considered external to the data warehouse since they are composed of operational data and the data warehouse has practically no control over it or it's format. Simply put, operational source systems are narrowly queried to get the necessary input data for the ETL process, meaning that the queries performed to retrieve the operational data don't have the extensive transformations that typically occur in the data warehouse.

ETL System

ETL stands for *Extract, Transform and Load*. This is the work area where the data preparation and transformation take place to build a data warehouse.

The first phase of the integration process is to successfully *Extract* the data from the source systems. Each data source can have different characteristics such as being from different database management systems, having different operating systems, hardware and communication protocols. All of this needs to be considered in order to adequately extract data into the ETL process, according to (R. Kimball & Caserta, 2004, pp. 55-111)

In other words, *Extracting* is the process of reading the source data and copying the required data into the ETL system, according to (Ralph Kimball & Ross, 2013 pp. 1-35). After the Extraction comes the *Transformation*, where many potential transformations may occur in order to cleanse the data of conflicts, missing records, duplicates, combining and standardizing data from multiple sources into standard formats, among other transformations designed to enhance the data and prepare it to be loaded into the data warehouse. That is the final step of the ETL System, to *Load* the data into the dimensions and fact tables of the dimensional model.

Within the ETL system there are many so called subsystems, as described by (Ralph Kimball & Ross, 2013, pp. 443-496), these are all processes that occur within the ETL system and can be divided in 4 different components:

- *Extracting*: As mentioned before it can be summarized as capturing the data from the source systems to be used in the ETL System. This process includes data profiling to understand and analyze the data being read from the source systems, changing the data capture system to make sure that in large amounts of data the data warehouse tables are only refreshed with the new data from the source systems, and lastly to accommodate the data load from multiple source systems. The source systems may greatly vary such as having data being extracted from multiple DBMS, XML sources, flat files, Web logs and even complex ERP systems. Some ETL tools may not even be able to support connections to load data from some legacy systems, in which case the users can use file extracts such as flat files to be ingested in the ETL system.

- Cleaning and conforming: Here is where the data is combined and transformed to be enhanced and conformed in standard formats. Duplicates need to be identified and removed and data quality needs to be assured.

- Delivering: This is where the data is prepared for presentation. The dimensions and fact tables are created and will compose the dimensional model responsible for populating the data in the presentation layer. Some of the subsystems that (Ralph Kimball & Ross, 2013, pp. 443-496) describes include creating slowly-changing dimensions, the recommended usage of surrogate keys in all dimensions, different fact table builders depending on the granularity of the data, and the usage of aggregates in order to improve performance.

- Managing: These are subsystems responsible for managing the ETL environment to ensure that the ETL processes are reliable in a sense that they must constantly run, are available at the right time to meet the SLAs, and manageable in terms of the data warehouse being able to constantly grow alongside the business requirements.

Presentation Area

The DW/BI presentation area is where the data is made available for the Business Intelligence applications to present to the end users. (Ralph Kimball & Ross, 2013, pp. 1-35) insist that the data must be presented in dimensional models, meaning either relational star schemas or OLAP cubes. This is also now widely accepted by the industry, that dimensional modeling is the most viable data warehouse technique to deliver the data to the business users.

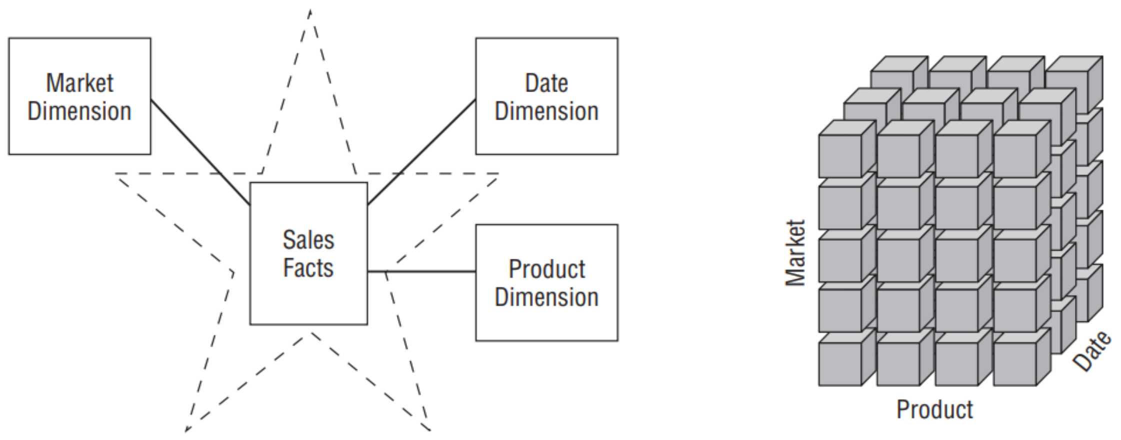


Figure 2 - Star Schema vs OLAP Cube (Ralph Kimball & Ross, 2013)

The presentation area must contain atomic data. This translates to the most granular level of data being in the fact table. This is to ensure that new business requirements can continuously be answered with the data in the dimensional model. Having conformed dimensions is very important because they allow for dimensional models to be combined and used together. According to (Ralph Kimball & Ross, 2013, pp. 1-35), the presentation area in a big DW/BI System ultimately consists of many dimensional models that share dimensions across the fact tables. This is the Bus Architecture of Ralph Kimball.

Business Intelligence Applications

Business Intelligence applications are tools and platforms used by the business users to read the data from the presentation area. A small percentage of business users may use ad hoc query tools to access the presentation area but most business users use platforms with already built dashboards and reports, which do not require them to create queries directly, and enable them to access the information in the final presentation form. These platforms can have an application of their own to access or even be browser-based to facilitate the access through the world wide web.

Independent Data Mart Architecture

This approach is also presented by (Ralph Kimball & Ross, 2013, pp. 1-35) and is used to suit departmental needs without the obligation to integrate the information that is found across the organization as a whole. A database is created from specific source systems to satisfy the departmental business requirements and needs, where specific labelling can be used. It's very common for other department to use the same data sources as the first department to answer their own business requirements. This results in multiple data marts being created, each emerging from their own department's requirements containing similar but more targeted solutions, as shown in Figure 3.

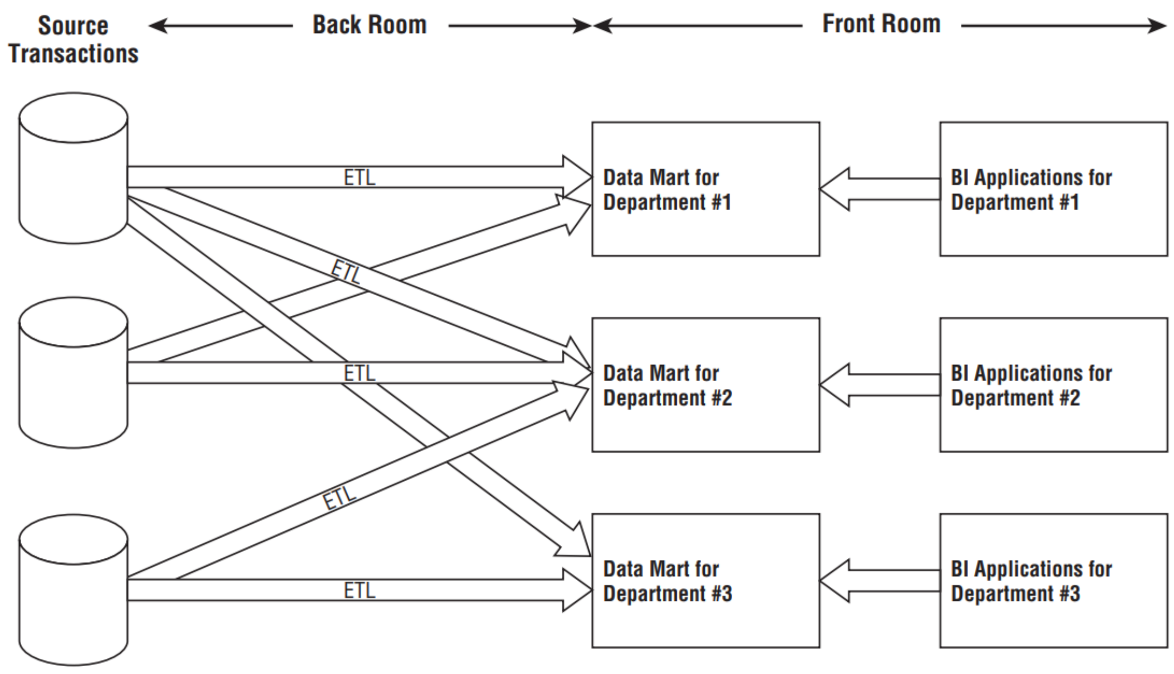

Source Transactions
Back Room
Front Room
ETL
ETL
ETL
ETL
ETL
Data Mart for Department #1
Data Mart for Department #2
Data Mart for Department #3
BI Applications for Department #1
BI Applications for Department #2
BI Applications for Department #3

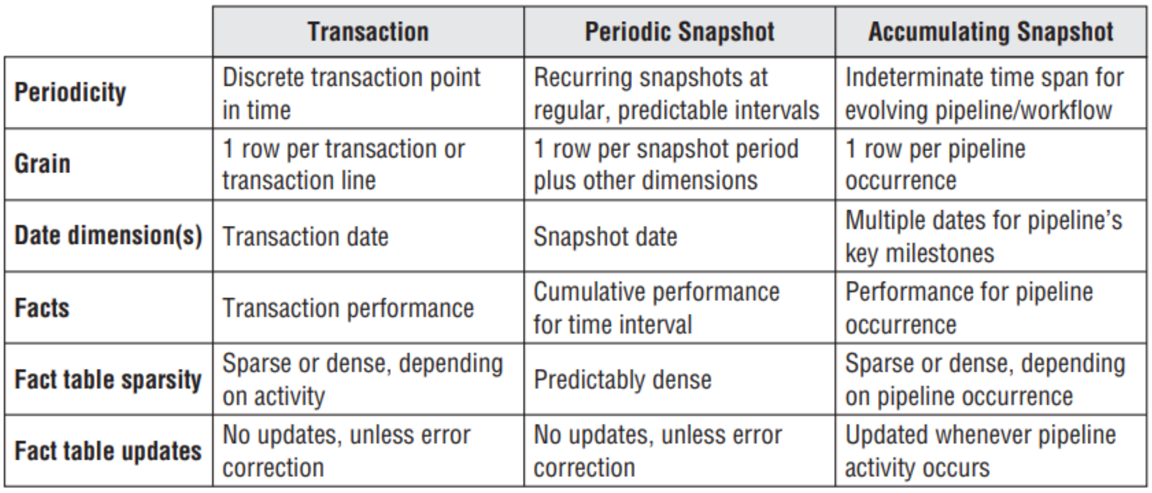
Figure 3 - Simplified illustration of the independent data mart "architecture"

The data in 2 data marts doesn't need to match because each data mart was constructed using the respecting department's business rules, which may differ.

This approach is predominant in the industry as many organizations start with independent data marts with no obligate coordination with the rest of the organization. It's also suited for fast development at a relative low cost. In the long run this approach can result in the need to reconciliate the data with the rest of the organization, which may pose some setbacks. Often this approach uses dimensional modelling because in this way the data is delivered in a manner that is easy to understand and fast to query. This architecture build is different from the Kimball DW/BI architecture because it is built around a department rather than the business process, which makes it harder to conform the dimensions at the organization level if needed, and many times it fails to use other core concepts of dimensional modeling such as focusing on the atomic data.

Types of Fact Tables

Facts tables are composed of the measures which are the critical measurements of an organization. They are the center of dimensional models. According to (Ralph Kimball & Ross, 2013, pp.473-475), there are three primary types of fact tables: Transaction, Periodic Snapshot and Accumulating Snapshot.



	Transaction	Periodic Snapshot	Accumulating Snapshot
Periodicity	Discrete transaction point in time	Recurring snapshots at regular, predictable intervals	Indeterminate time span for evolving pipeline/workflow
Grain	1 row per transaction or transaction line	1 row per snapshot period plus other dimensions	1 row per pipeline occurrence
Date dimension(s)	Transaction date	Snapshot date	Multiple dates for pipeline's key milestones
Facts	Transaction performance	Cumulative performance for time interval	Performance for pipeline occurrence
Fact table sparsity	Sparse or dense, depending on activity	Predictably dense	Sparse or dense, depending on pipeline occurrence
Fact table updates	No updates, unless error correction	No updates, unless error correction	Updated whenever pipeline activity occurs

Figure 4 - The fact table builders (Ralph Kimball & Ross, 2013, pp. 111-139)

Transaction fact tables, as explained by (Ralph Kimball & Ross, 2013, pp. 119-122 & 473-475) represent events that occurred in a particular moment in time, meaning that each row in the fact table represents a transaction. They are typically the largest fact tables of the three in terms of volume of data. The atomic data present is usually bound to the daily grain or in some cases complemented with the time stamp, depending on the requirements of the business.

The Periodic Snapshot fact table is loaded in a similar fashion as a transaction fact table, but follows a regular repeating measurement in a period of time. In other words, snapshots are taken periodically of account balances, monthly financial reporting and bank account monthly statements, for examples. These periods must regular and can be defined daily, weekly or monthly, those are the most common periods. These periods are defined in a single date column.

Accumulating snapshots have a different load process than the previous two. They are designed to represent the evolving status of a process, meaning that they can have two or even five date columns, each describing an occurrence in the process flow. Some of these date columns can be exemplified by an order processing and have many date columns such as order date, shipping date, payment date and delivery date. Some of these date columns may not have a date when the order was placed and other columns may need to be revisited and updated. This means that each time something happens, the fact table rows may be completely modified.

Of the 3 fundamental fact tables, the OLAP cubes support transaction and periodic snapshot fact tables but cannot work with accumulating snapshot fact tables because of certain constraints that have to do with the overwriting of the facts (Ralph Kimball & Ross, 2013, pp. 1-35).

Another approach introduces an effective date and a termination date in the fact table, making it a temporal fact table, as presented by (Mauri, 2012). This originated primarily to solve the issue with having fact tables with an enormous number of snapshots. Rather than having to take daily snapshots for current and future changes that can go on for years, by having a date range, a single record contains relevant data referring to a period of time, while only introducing new records when changes occur. This type of fact table can be used in similar scenarios, such as in pricing, where discounts have effective and expired dates.

Kimball's Four-Step Dimensional Design Process

According to (Ralph Kimball & Ross, 2013, pp. 37-68) there are 4 key steps in designing a dimensional model:

1. Selecting the business process
2. Declaring the grain
3. Identifying the dimensions
4. Identifying the facts

The operational activities that occur in an organization define the business process. Some examples of these activities or events are generating orders, making appointments, phone calls or capturing account balances every month. Understanding what is happening in a business process will convert to the key metrics that generate facts that will be used in a fact table. Every activity in a business process will translate to a row in a fact table. This is the first step in designing a dimensional model, because once the business process is defined it allows for the grain, dimensions and facts to be determined.

The next step is the most important in designing a dimension model. Declaring the grain is the same as declaring exactly what a single row in a fact table stands for. The higher the grain the more detailed the data is in a fact table. An atomic grain is the lowest grain possible which represents the highest detail in a fact table. By using an atomic grain, the fact table is protected against future business needs that may require a high level of detail. Each fact table must have one level of granularity, in case different granularity is found in two business processes within the same organization, then each business process will have their own fact table because granularities cannot be combined in the same fact table.

After the grain has been declared, the dimensions can be identified. They provide the context for the facts. In other words, they provide the descriptions for the business processes. The "who, what, when, where, why and how" are all questions that are answered with the attributes in the dimension tables. The relation between the dimension and fact table must always be one-to-many. This is to say that, for example, one customer can have many transactions but each transaction can only be linked to one customer. All the surrounding attributes of that customer such as first and last name, address and country can be used to form the dimension customer.

Lastly, the facts can be identified. Facts are measurements that result from business process activities. They are defined at the same level as the grain. For example, if a business process is characterized by online sales, then the sales amount and quantity sold are facts to be considered in the fact table.

Dimensional Modeling Techniques

(Ralph Kimball & Ross, 2013, pp. 37-68) has many dimensional modeling techniques which have been accepted by the industry as best practices.

All of Kimball's dimensional modeling techniques should be considered when building a dimensional model. Some relevant examples of techniques to be considered for this project are highlighted in this section.

Conformed Facts

The same fact can be present in more than one fact table. When this happens, the fact is called a conformed fact. Special attention must be taken to ensure that the facts are indeed identical. They should have the same name when they are conformed facts, but if that's not exactly the case, different naming should be used in order to avoid confusion.

Nulls in Dimension and Fact Tables

Contrary to dimension tables, where nulls should not exist and be substituted by inferred members such as "Unknown" or "Not Applicable", nulls in fact tables are acceptable in measures because the aggregate functions behave well with nulls. Foreign keys and surrogate keys, however, should not be allowed nulls as this can violate the referential integrity with the dimension tables.

Dimension Surrogate keys

Dimension tables must have a unique primary key, which is a column that defines the highest granularity in a dimension. For the DW/BY system to better administer control over the dimensions, rather than using the natural key in each dimension, surrogate keys are created for the dimensions, which are better suited, for example, in cases where a dimension's natural key changes are tracked over time and generate multiple rows. Surrogate keys are integer numbers that start at number 1 and are incremented in a sequence every time a new dimension row is added. The Date dimension is excluded from this surrogate key rule. Applying an integer key in the YYYYMMDD format provides more meaning to the Date dimension and is more easily applied and understood in the fact table.

Degenerate Dimensions

Degenerate dimensions are dimensions that are only composed of the primary key and no other columns. Because the dimension only has one column, it's unjustified to create a dedicated dimension which will not bear further descriptive columns. In these cases, the column can be incorporated directly into the fact table. Common examples of these columns are invoice numbers and ID numbers such as transaction IDs. Degenerate dimensions are most commonly found in transaction and accumulating snapshot fact tables.

2.2.1.2. Inmon's Approach

Data warehouse evolution and progress

(W.H. Inmon, Derek Strauss, 2008, pp. 1-22) describes the various reasons and needs that led the progression of the early stages of the data warehouse to the more complex and elaborated data warehouse architecture commonly observed today, as illustrated in Figure 5.

According to (W.H. Inmon, Derek Strauss, 2008, pp. 1-22), this evolution took place as a result of the upgrade of technologies and their ability to communicate information to the end user, the emergence of online processing, the need for having integrated corporate data, the need for integrating raw, textual and unstructured data, the increase in speed and capacity of technology and the unit cost reduction of technology, which has been coming down over the years.

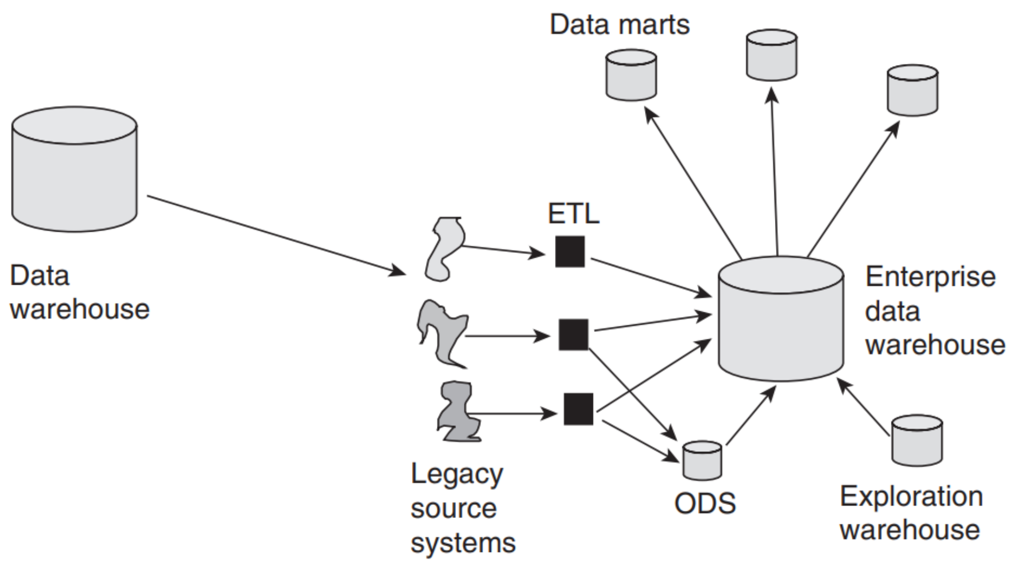


Figure 5 - Data Warehouse evolution to a full blown architecture (W.H. Inmon, Derek Strauss, 2008, pp. 1-22)

The components of the full-fledged data warehouse:

Extract, Transform and Load, as described by (W.H. Inmon, Derek Strauss, 2008, pp. 1-22), is the process that enables the extraction of the data from the legacy source systems, which can be unstructured and raw and transform it into corporate data. The ETL is responsible for fulfilling many functions such as conversion of data, creating default values when applicable, summarizing data, adding time values on key data, restructuring data, merging data and deleting redundant data.

ODS stands for operational data store. It's where the online update of the integrated data takes place using OLTP and can be found in a standardized format. After integrated, the ODS data is available for processing and updating with high performance.

The data mart component of the architecture is where the ends users are able to access the data and make appropriate analysis of the data that they want to see. In most cases the data mart is created using a different technology and not holding as much data as the data warehouse. The data mart holds summarized and aggregated data.

The exploration warehouse objective is to enable end users to do exploration analysis on the data, in most cases while the project is ongoing and not complete.

Hub-and-spoke Corporate Information Factory architecture

The Hub-and-Spoke Corporate Information Factory (CIF) architecture is recommended by Bill Inmon. The illustration in Figure 6 is a simplified version described by (Ralph Kimball & Ross, 2013, pp. 1-35) and highlights the main components behind this architecture.

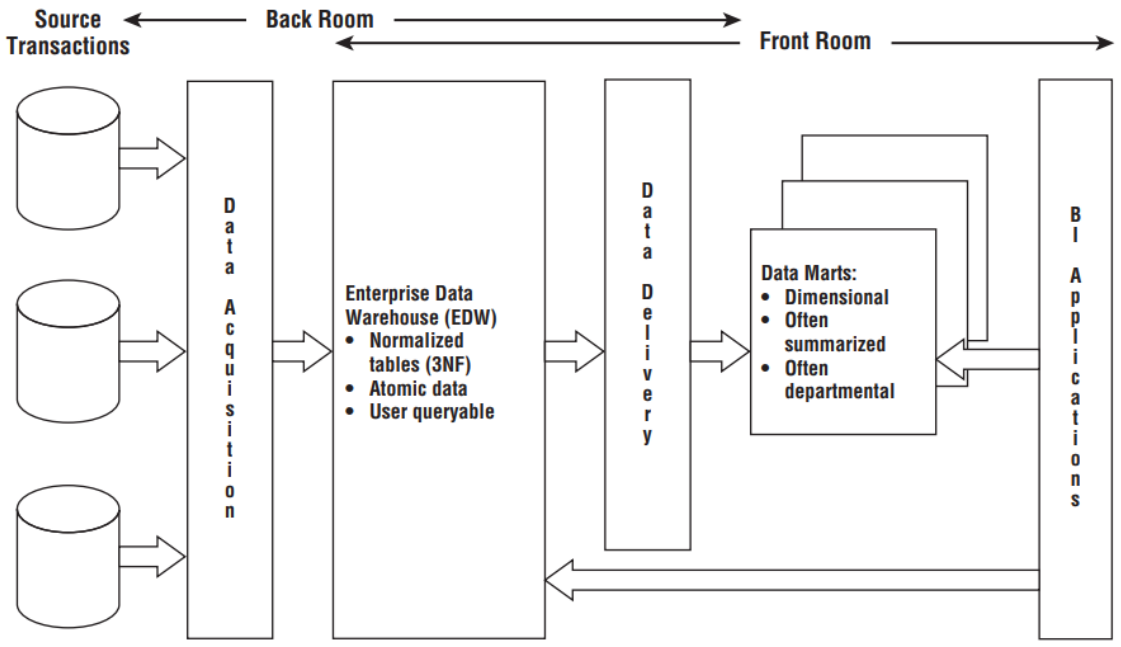


Figure 6 - Simplified illustration of the Hub-and-spoke Corporate Information Factory architecture (Ralph Kimball & Ross, 2013)

The first step is called Data Acquisition and refers to the data being extracted from the operational source systems. The ETL process then uses the resulting atomic data to populate the Enterprise Data Warehouse (EDW), which is in the third normal form (3NF). Business Intelligence applications then consume the data either directly from the EDW or from data marts that read the data from EDW and are often summarized, departmental and dimensionally structured.

2.2.1.3. Differences between Inmon's and Kimball's Approaches

According to (Turban et al., 2011, pp. 29-78), the overall difference between Inmon and Kimball's approach is that Inmon's uses a Top-down approach while Kimball uses a Bottom-up. While Inmon builds the Enterprise Data Warehouse first and only builds data marts on top of that, Kimball's approach focuses on the data marts for faster business delivery which can later then be integrated into a data warehouse.

Kimball's data mart bus architecture with conformed dimensions model a business process whilst Inmon's hub-and-spoke architecture (e.g. Corporate Information Factory) is frequently departmentally-centric, as stated by (Ralph Kimball & Ross, 2013, pp. 1-35).

(Turban et al., 2011, pp. 29-78) defines Inmon's model to be quite complex and Kimball's to be fairly simple. In fact, (Ralph Kimball & Ross, 2013, pp. 1-35) argue that the most complex Corporate Information Factory is unworkable as a data warehouse.

Kimball's data warehousing tools follow dimensional modeling techniques that depart from relational modeling while Inmon's approach applies more traditional tools such as ERD and data flow diagrams.

The primary audience of the Kimball DW/BI architecture is directed at end users. Inmon's Enterprise Data Warehouse is meant to be used by IT professionals.

The main objective of Kimball's approach is to deliver a solution that is easy to query for the end users with reasonable performance. Inmon, on the other hand, focuses on delivering a solid technical solution based on proven database methods.

2.2.2. Presentation Layer

According to (Few, 2006, pp. 11-28), a dashboard consists on the display of an agglomeration of the most important information needed to achieve one or more objectives. The information is presented in a visual manner and consists of both textual and graphical data, and so it can contain metrics, KPIs, tabular data, charts, filters, drilling-down options and can have a static or a dynamic interface. A dashboard must be concise, clear, and intuitive and must be tailored to the requirements of a specific person or group and display the information needed to achieve specific objectives. All the information in a dashboard must fit on a single screen. The purpose is to deliver the necessary information without exerting much effort and time on the user.

(Few, 2006, pp. 29-33) categorizes dashboards into three main roles: Strategic, Analytical and operational. The dashboard design must, therefore, adopt the appropriate role based on the business requirements and needs. The primary use of dashboards today is to present strategic data. These dashboards provide quick overviews to help monitor the help and opportunities for the business. They focus on high-level measures and have usually very simple and direct displays, therefore, not designed for detailed interactions and analysis. These dashboards work well with snapshot data that can be loaded monthly, weekly and daily. Analytical dashboards support analysis for the end users. With these dashboards, interactions, comparisons and the ability to drill-down and look at more detailed data are functionalities that are present to aid in analysis. As an example, when sales are decreasing, more information must be present in the dashboard to understand and discover patterns and reasons to explain the behavior. These dashboards also benefit from data loads that occur from time to time, being it monthly, weekly or daily. Operational dashboards have the purpose of monitoring operations. This means that changes in activities that require immediate attention must be displayed instantly on the dashboard. Contrary to the strategic and analytical dashboards, the operational dashboard requires real-time data in order to tackle an issue as soon as it occurs.

As stated by (Few, 2006, pp. 38-62), there are many aspects that can differentiate a good dashboard from a poorly designed one. Chart placement, appropriate space usage, structure, color palette, relevant chart combinations, relevant metrics and KPIs, proper chart scaling, proper chart highlights, among others greatly influence the attractiveness and efficiency of a dashboard.

(Few, 2006, pp. 80-94) presented a layout placement where certain parts of the dashboard are given more attention than others mainly due to the conventions of most western languages. According to him, the central part of the dashboard and the top left corner are more emphasized and should present the most important information. The top right corner and bottom left corner are neither emphasized nor de-emphasized and the bottom right corner is de-emphasized and should present the least relevant information.

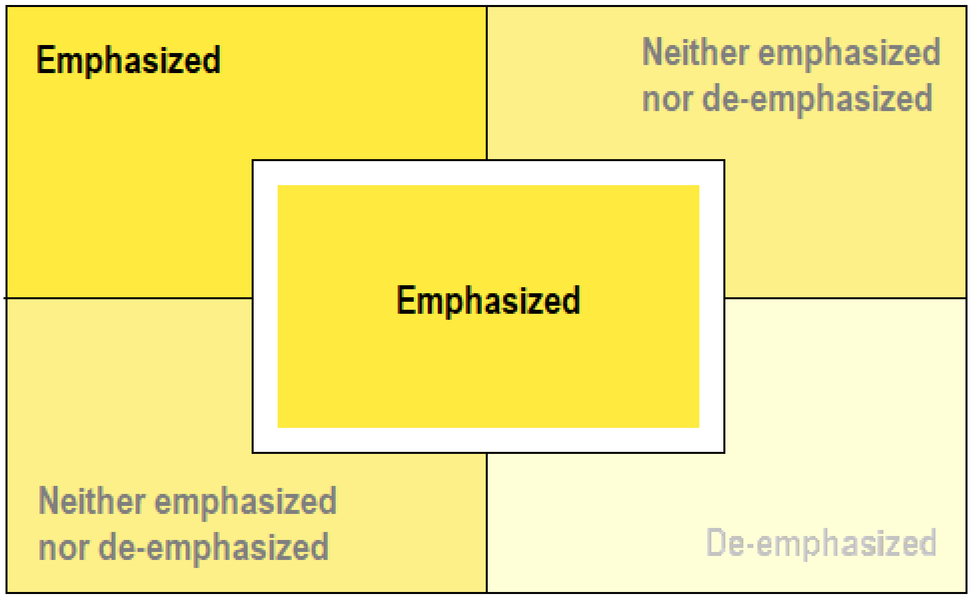

Emphasized
Neither emphasized nor de-emphasized
Emphasized
Neither emphasized nor de-emphasized
De-emphasized

Figure 7 - The different degrees of visual emphasis that are associated with the different regions of the dashboard

Key Performance Indicators (or KPIs) are a type of indicator that measures the progress of a certain metric towards its goal as described by (Shahin & Mahbod, 2007). And as these authors state, a SMART (Specific, Measurable, Attainable, Realistic, Time-sensitive) criteria can be used for goal setting in a company. An AHP (Analytical Hierarchy Process) methodology is used to structure a problem into a hierarchical framework. As KPIs are derived from the company goals, a combination of SMART goal setting criteria and AHP can determine which KPIs are more relevant to the company's goals.

According to (Larson, 2016, pp. 13-21), Business Intelligence is useful at all levels of an organization, but the information delivered should be different according to the managerial hierarchy for efficient decision-making. The hierarchy is divided into a 3-level pyramid: Top-level, mid-level and broad base.

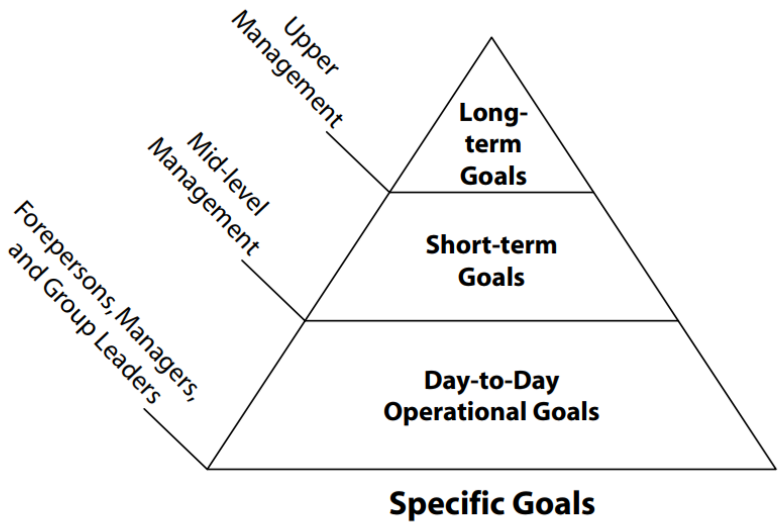


Figure 8 - Specific goals at each level of the organization

At the top-level management, it’s important for the decision makers to get the big picture of the organization and not get lost in the details, since they are responsible for the long-term goals.

At the middle layer of the pyramid, the decision makers are responsible for setting short-term goals and operate at the department level within the organization.

At the broad base, the decision makers consist of business managers dealing with day-to-day operations and are responsible for allocation the resources for the next shift or the next sales campaign, for example. These users need a detail view of the business process, meaning the highest level of detail and also responsiveness in accessing the data.

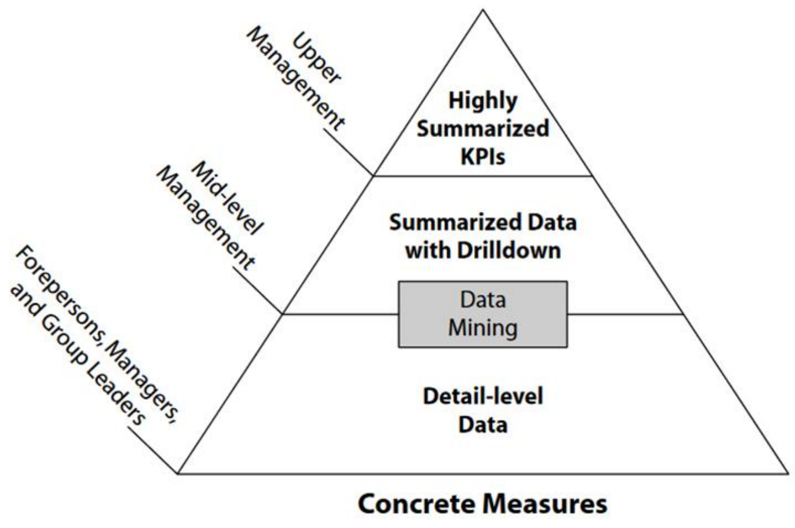


Figure 9 - Concrete measures at each level of the organization

The granularity of the business intelligence delivered to the top-level decision makers must be low to better accommodate the effectiveness of the decision making at this level. The measures presented to the upper management must, therefore, be highly summarized. Many times, these measures are illustrated by indicators that classify the measure according to its value inserted within a range. This range will indicate if that particular measure is performing well or not and label the result accordingly. These indicators are also known as Key Performance Indicators.

Business users at the mid-level management also benefit from summarized data but many times need to drill-down to access more detailed data.

On the base of the pyramid, forepersons, managers and group leaders at this level need to get business insights from accessing the data at the operational level. In some cases, they can benefit from summarized measures but need a drill-down mechanism to access the highest granularity of detail.

3. METHODOLOGY

The methodology that was used for this project is Kimball's data mart bottom-up approach, considering that the business requested a fast delivery departmental solution that can later be integrated in a data warehouse where the fact tables share conformed dimensions.

To start designing Kimball's four-step dimensional design, because there are two different business processes, there must be two separate designs. This topic will be addressed in the architecture in subchapter 3.2, after the data sources have been carefully analyzed. The following subchapters will also address the main ETL processes in the landing, staging area, data warehouse and presentation layer.

The database management system and ETL tool used in this project are from Microsoft. As such, SQL Server Management Studio and SQL Server Integration Services were used to create the tables, indexes, and all the transformations that happen in the Business intelligence Architecture Framework adapted from the back room of Kimball's core elements of a data warehouse (Ralph Kimball & Ross, 2013, pp. 1-35). The illustration of the adapted version of the architecture framework will be presented in subchapter 3.2.

The Business Intelligence application that was used to access the data from data marts to the presentation layer is the Qlik Sense data visualization tool (Qlik, 2019). This tool has been chosen to present the final results of this project because it is the standard reporting tool used by the business users that also use it for other departments. This tool has a wide variety of features to present the information in multiple ways with easy to navigate dashboards. This enables us to present the results in interactive visualizations and access drill-through features with a simple mouse click. Careful dashboard design was conducted for the pricing managers to be able to get the most important business insights between the sales and the respective discounts in a clear and concise manner.

SQL Server Analysis Services was not used because Qlik Sense does not support the loading connection to OLAP Cubes. Qlik Sense associative engine has an in-memory fully index nature, which includes compressed binary indexing, logical inference and dynamic calculation. Users can create hierarchies on the fly, without having to pre-aggregate data in advance (Qlik, 2020). Slicing and dicing happen with almost all selections and recalculations take place almost instantly. In essence, Qlik Sense has an associative model of its own that replaces the necessity of an OLAP Cube.

The biggest challenge, however, as mentioned before, are the ETL transformations and methodology that was created specifically to overcome the challenge with the overlapping of date ranges that will be covered in great detail in subchapter 3.3.2.2, 7.3, and 7.4. As such, the main focus of the ETL process will be around these advanced pricing transformations.

3.1. DATA SOURCES

The data sources provided that will be used as input for this project are comprised of two excel files: one consisting of the oracle advanced pricing agreements and details regarding rules, customers, products and list prices and the other file consisting of the transactions that followed the advanced pricing agreements. The first export describes the advanced pricing list of products, customers, list prices, discount rebates, special discounts, date effective, date expired, among other attributes for each pricing rule. The second export is transactional and presents the invoiced sales that occurred with the discounts applied from the advanced pricing agreements list.

3.1.1. Advanced Pricing Agreements Data

The advanced pricing agreements export contains the following excel sheets:

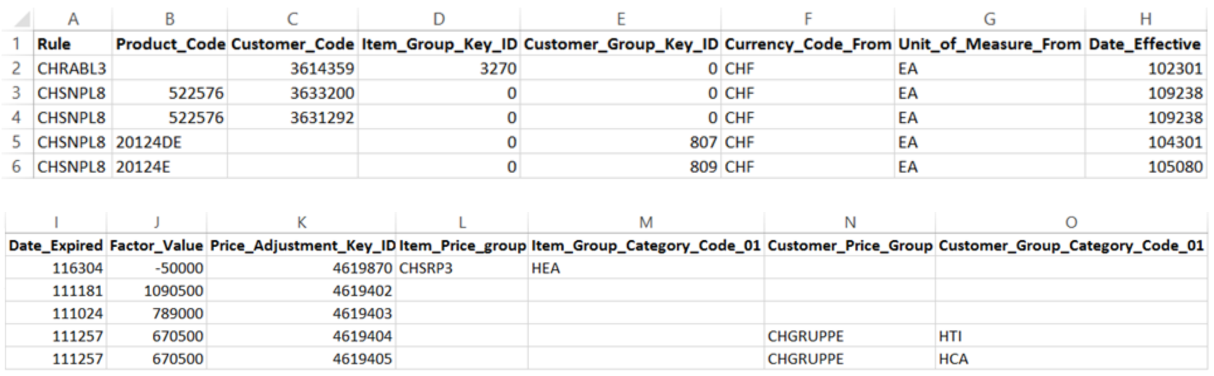
3.1.1.1. PriceRules

This sheet contemplates a list of the advanced pricing agreements. Each record represents an agreement that is made to a customer or customer group, to a product or a product group, the time period that the agreement takes effect and expires, the code of the rule that is being applied and other attributes such as the currency, factor value and the attributes that specify the link to the customer group and product group.

The factor value is the discount that is being applied. This column contains two types of discounts: Special Prices and Discounts in percentage. The values are all automatically multiplied by 10.000, so to get the real value they must be divided by 10.000. When the value is negative, it refers to the discount in percentage that is being applied. For example, the first record holds a factor value of -50.000. This means that the discount being applied is 5%. When the value is positive, it refers to a special fixed price. For example, the second record has a factor value of 1.090.500. This means that this agreement offers a special price of 109,05 CHF for that product and customer within that period of time.

The dates that indicate when the discount takes effect and expire are in JDE Julian Date format. This date is in the format YYYddd, where the Year starts at 1900 and the days are summed up as an integer. For example, the first record has a date_effective of 102301. To convert this to Gregorian date to get a better understanding of the actual date we first separate the year (YYY) from the day (ddd). The 102 is the year 1900 + 102 = 2002 and the day 301 is the 301st day of that year, meaning the 28th of October.

The columns Item_Price_Group and Item_Group_Category_Code_01 are used to link the product groups to their respective product codes following a logic that will be explained in detail in subchapter 3.3.2.1, whenever the agreement is done at the product group level and no individual product code is specified. The same is true for the columns Customer_Price_Group, and Customer_Group_Category_Code_01 for the customer groups whenever the customer code is empty.

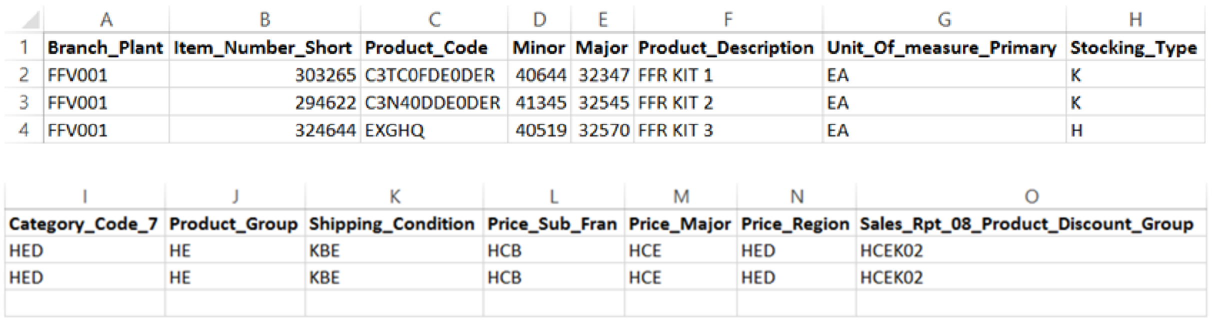


	Rule	Product_Code	Customer_Code	Item_Group_Key_ID	Customer_Group_Key_ID	Currency_Code_From	Unit_of_Measure_From	Date_Effective	Date_Expired	Factor_Value	Price_Adjustment_Key_ID	Item_Price_group	Item_Group_Category_Code_01	Customer_Price_Group	Customer_Group_Category_Code_01
2	CHRABL3		3614359	3270	0	CHF	EA	102301	116304	-50000	4619870	CHSRP3	HEA		
3	CHSNPL8	522576	3633200	0	0	CHF	EA	109238	111181	1090500	4619402				
4	CHSNPL8	522576	3631292	0	0	CHF	EA	109238	111024	789000	4619403				
5	CHSNPL8	20124DE		0	807	CHF	EA	104301	111257	670500	4619404			CHGRUPPE	HTI
6	CHSNPL8	20124E		0	809	CHF	EA	105080	111257	670500	4619405			CHGRUPPE	HCA

Table 1 - PriceRules Data Source Sample Records

3.1.1.2. Product

The Product sheet is comprised of the list of product codes and the descriptive attributes for each product, including the columns for the Price_Sub_Fran, Price_Major, Price_Region and Sales_Rpt_08_Product_Discount_Group that define the logic behind the product groups that will be discussed in the subchapter 3.3.2.1.

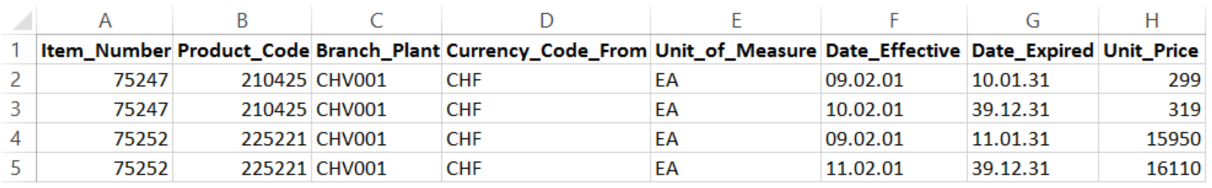


	Branch_Plant	Item_Number_Short	Product_Code	Minor	Major	Product_Description	Unit_Of_measure_Primary	Stocking_Type	Category_Code_7	Product_Group	Shipping_Condition	Price_Sub_Fran	Price_Major	Price_Region	Sales_Rpt_08_Product_Discount_Group
2	FFV001	303265	C3TC0FDE0DER	40644	32347	FFR KIT 1	EA	K	HED	HE	KBE	HCB	HCE	HED	HCEK02
3	FFV001	294622	C3N40DDE0DER	41345	32545	FFR KIT 2	EA	K	HED	HE	KBE	HCB	HCE	HED	HCEK02
4	FFV001	324644	EXGHQ	40519	32570	FFR KIT 3	EA	H							

Table 2 - Product Data Source Sample Records

3.1.1.3. ListPrice

The ListPrice sheet contains the information regarding the list prices for each product code, the currency, unit of measure they are sold in and the periods that the various list prices are in effect. The column Unit_Price refers to the List_Price in this sheet and will, therefore, be labeled as such.

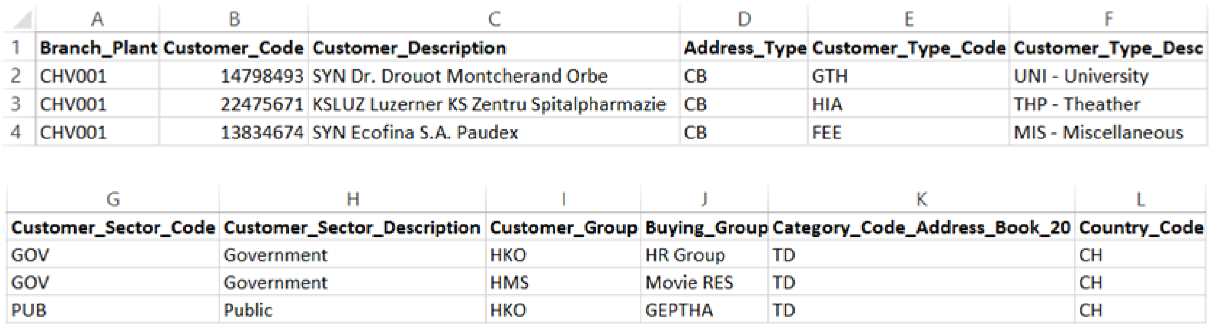


	Item_Number	Product_Code	Branch_Plant	Currency_Code_From	Unit_of_Measure	Date_Effective	Date_Expired	Unit_Price
2	75247	210425	CHV001	CHF	EA	09.02.01	10.01.31	299
3	75247	210425	CHV001	CHF	EA	10.02.01	39.12.31	319
4	75252	225221	CHV001	CHF	EA	09.02.01	11.01.31	15950
5	75252	225221	CHV001	CHF	EA	11.02.01	39.12.31	16110

Table 3 - ListPrice Data Source Sample Records

3.1.1.4. Customer

The Customer sheet, much like the product sheet, holds the relevant attributes that describe each customer code, such as the address type, customer type code, customer sector, buying group, country code and the Customer Group.



	A	B	C	D	E	F	G	H	I	J	K	L
1	Branch_Plant	Customer_Code	Customer_Description	Address_Type	Customer_Type_Code	Customer_Type_Desc	Customer_Sector_Code	Customer_Sector_Description	Customer_Group	Buying_Group	Category_Code_Address_Book_20	Country_Code
2	CHV001	14798493	SYN Dr. Drouot Montcherand Orbe	CB	GTH	UNI - University	GOV	Government	HKO	HR Group	TD	CH
3	CHV001	22475671	KSLUZ Luzerner KS Zentru Spitalpharmazie	CB	HIA	THP - Theather	GOV	Government	HMS	Movie RES	TD	CH
4	CHV001	13834674	SYN Ecofina S.A. Paudex	CB	FEE	MIS - Miscellaneous	PUB	Public	HKO	GEPTHA	TD	CH

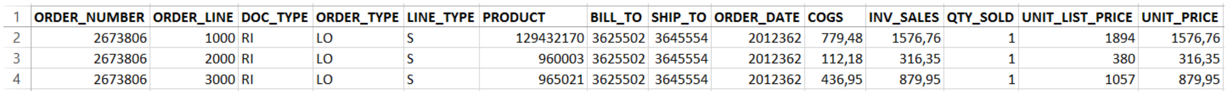
Table 4 - Customer Data Source Sample Records

3.1.2. Transactions Data

The excel export contains only one sheet with transactional data. Examples of transactions are illustrated in Table 5.

3.1.2.1. Invoiced Sales

Each record represents an invoiced sale that resulted from the advanced pricing agreements and is accompanied by more details such as the order number, order line, order type, line type, the product code and the customer code, which is the column represented by the bill_to, the ship_to customer, order_date, cost of goods sold (COGS), the unit list price, the final unit price, the quantity sold, and the invoiced sales.



1	ORDER_NUMBER	ORDER_LINE	DOC_TYPE	ORDER_TYPE	LINE_TYPE	PRODUCT	BILL_TO	SHIP_TO	ORDER_DATE	COGS	INV_SALES	QTY_SOLD	UNIT_LIST_PRICE	UNIT_PRICE
2	2673806	1000	RI	LO	S	129432170	3625502	3645554	2012362	779,48	1576,76	1	1894	1576,76
3	2673806	2000	RI	LO	S	960003	3625502	3645554	2012362	112,18	316,35	1	380	316,35
4	2673806	3000	RI	LO	S	965021	3625502	3645554	2012362	436,95	879,95	1	1057	879,95

Table 5 - Invoiced Sales Data Source Sample Records

3.2. ARCHITECTURE

The architecture sets the foundation designs on how the data will be organized from beginning to end. As such, this subchapter will cover the dimensional modeling and the stages of the ETL Process that take part in defining the Business Intelligence Framework.

3.2.1. Kimball's 4-step Dimensional Design Process

To start designing the dimensional model according to (Ralph Kimball & Ross, 2013, pp. 37-68), the first step is to select the business process. There are two separate business processes that complement each other: advanced pricing agreements and invoiced sales. Nonetheless, because the operational activities are different, their dimensional designs must be handled separately and later joined through conformed dimensions.

3.2.1.1. Advanced pricing Agreements

The advanced pricing agreements follow a business process directed by pricing managers. They are responsible for adjusting the prices and discounts from time to time. These activities result in a series of personalized discounts that are valid for the customers in a certain period of time. The discounts can stack, which decreases the overall final price of the products. As such, the grain is declared as the discounts that are effective in a certain period of time for a particular customer and product. This is what a single row will represent in the fact table for the advanced pricing agreements, because it will enable to understand which discounts were applied that resulted in a specific final price, which is a detail that is not visible in the invoiced sales export.

The next step is to identify the dimensions that will provide the contextual descriptions for each advanced pricing agreement grain. For each row in the fact table, there must be a link to the customer and product that are having the APA, the effective and expiration dates of these agreements, the discount rules that are being applied within this period, and the currency that is being practiced.

The final step comes down to the facts that will take part in each row of the grain. These facts are comprised of the list price, the multiple discounts that can possibly occur within the same period and the final price. This final price will be calculated based on the list price and the business logic that was instructed by the business analyst when the exports were provided for this project. As such the final price in the advanced pricing fact table should be the same as the final price that will be discussed below in the transactional dimensional model. This final price cannot be guaranteed to be the same in both fact tables because in the APA fact table it is calculated, while in the transactions fact table it comes from the source. However, in the reporting layer these two measures can be compared and conformed and in case they don't match, they can be directed to a data quality report for the business users to be aware of the differences.

3.2.1.2. Transactions

Transactions are a different type of business process that is responsible for registering invoiced sales and sending the products to the customers. The atomic data is represented by a daily transactional grain, which translates to each row in the fact table being a transaction recorded for a customer and product at a particular date.

Now with the business process and grain established, the next step is to define the dimensions. Similar to the advanced pricing agreements, the transactions business process also shares some of the same dimensions. In other words, each transaction needs to identify the customer and product, the date that the transaction happened, and the currency in which it was held.

The fourth and final step is to define the facts for the fact table. They are COGS, Invoiced_Sales, Quantity_Sold, Unit_list_Price, which represents the list price, and Unit_Price, which is the same as the final price for that transaction.

3.2.2. Business Intelligence Framework

The ETL processes follow the principles of (R. Kimball & Caserta, 2004, pp. 247–253) and can be separated into three stages: Landing (exact copy of the data source), Staging Area and Data Warehouse. Using Microsoft SQL Server Data Tools (Microsoft, 2013), The two data sources were carefully analyzed and modeled in two ETL processes: one for advanced pricing and another for transactions, but the main focus of this project report is on the advanced pricing agreement's model since it holds the major developments in the transformational layer.

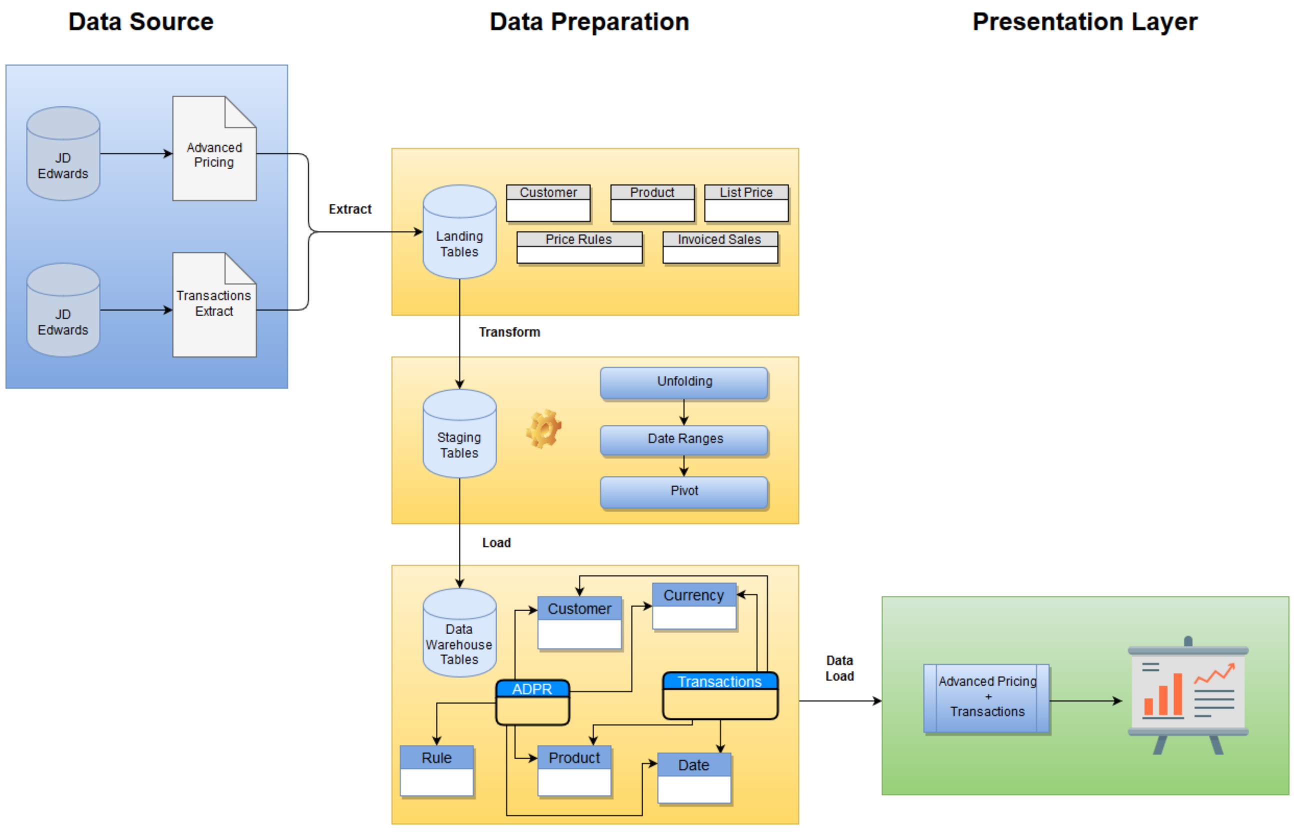


Figure 10 - Business Intelligence Architectural Framework Adaptation

Figure 10 shows an adaptation of the Business Intelligence Architectural Framework by (R. Kimball & Caserta, 2004, pp. 29–52). At the Landing level, the data is simply copied from the original source, which is the same as in the Microsoft Excel file exports, into a database named ADPR_Landing in Microsoft SQL Server Management Studio. No significant changes occur to the data in this process other than renaming the attributes and measures to relevant business names. At the SA, the data will be extracted from the Landing and transformed using various methods and modelling tools into a database named ADPR_Staging. Figure 10 only shows 3 levels of transformation because these are the main focus of this project. All other transformations are more common and are not exhibited for redundancy reasons. At the DWH level, the data is loaded from the SA to the database ADPR_DWH and each business process is developed into a Star Schema dimensional model (R. Kimball & Caserta, 2004, pp. 9–20), which combined through the conformed dimensions form a Fact Constellation Schema.

Once the data management of the business intelligence architectural framework is completed (R. Kimball & Caserta, 2004, pp. 16–20) which in our adaptation figure is the combination of the Data Source and Data Preparation, the data from both fact tables will be combined and loaded into the presentation layer in a way that enables the creation of the dashboards in Qlik Sense (Qlik, 2019), holding the information that empower the pricing managers with the necessary key insights to make more targeted pricing adjustments.

3.3. ETL Processes

The ETL process that follow adhere to Kimball's dimensional modeling techniques and aim to construct independent data marts that present data for the business users that can later be integrated in a bottom-up approached data warehouse.

3.3.1. Landing

The ETL processes behind the landing transformations are very straightforward. They reflect a copy of the data from the excel files with some label changes and minimal to no transformations.

An example of the SSIS package for the landing is shown below in figures 11 and 12 for the customer:

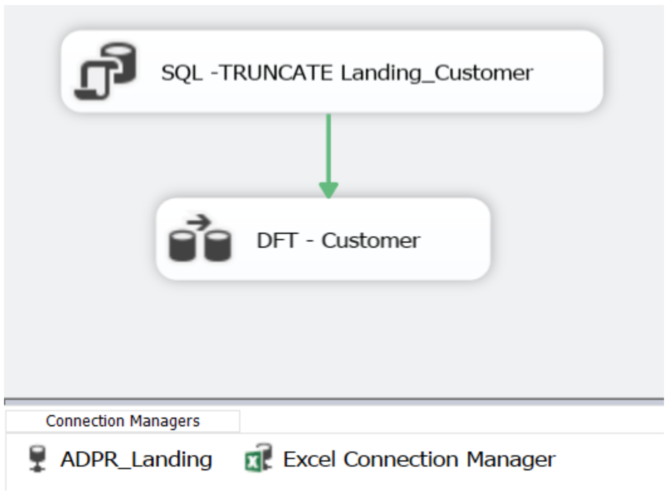


Figure 11 - SSIS Control Flow - Customer landing

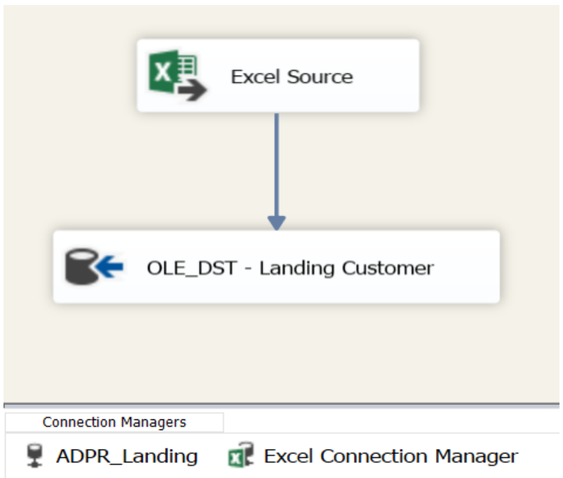


Figure 12 - SSIS Data Flow - Customer Landing

As such, the first layer of the ETL, which extracts the data from the excel files and populates the landing tables consist of the SSIS packages that are named:

- E_Price_Rules.dtsx
- E_Customer.dtsx
- E_Product.dtsx
- E_ListPrice.dtsx
- E_Invoiced_Sales.dtsx

3.3.2. Staging Area

On the Staging Area, the three main transformations displayed in the Business Intelligence Architectural Framework adaptation focus on the advanced pricing agreements. Since they are the essence of this project report, for each transformation there will be a simplified theoretical example for contextual purposes followed by a real example. The full details of these transformations and other less relevant transformations can be found in the attachments on chapter 7.

The landing of the invoiced sales can easily be modeled into a fact table. Hence, the transformations in the staging area are fairly straight forward.

For the Advanced Pricing business process, the objective is for the grain to be a record for each combination of customer and product, showing all the discounts that are being applied within the same time frame. Since we have the list price and the discounts, the final price can be calculated and incorporated into the fact table. The transformations on the price rules of the advanced pricing agreements model are divided into three stages:

- Unfolding
- Date Ranges and List Price
- Pivot

3.3.2.1. Unfolding

Unfolding is the name given to the first transformation of the data that comes from the advanced pricing agreements from the landing schema to the staging table staging.PriceRulesUnfolding.

The products can be defined at the product code or the product group level. A Product Group is a list of product codes. The same logic applies to the customers. Whenever a product code or a customer code is missing, it means that the agreement was made at the respective group level.

In the landing tables, each advanced pricing agreement can be applied to the product group and also customer group. So, for the first staging transformation, the objective is to have a combination of product code and customer code for every advanced pricing agreement by unfolding every product group and customer group into their respective product codes and customer codes.

Simplified Theoretical Example

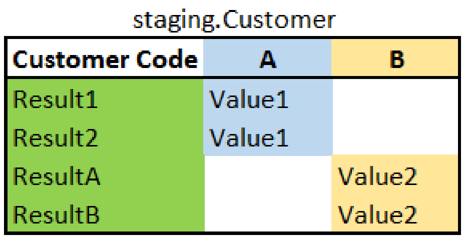
Using a simplified version of the landing.PriceRules (table 6), staging.Customer (table 7) tables and staging.PriceRulesUnfolding (table 8), the goal is to query the first two tables to achieve the latter table as a result. Since this example does not need to look up the product groups, the table staging.Product is not considered.

landing.PriceRules

Product Code	**Customer Code**	**Product Price Group**	**Product Category Code**	**Customer Price Group**	**Customer Category Code**
Prod01	aaaa				
Prod02	bbbb				
Prod03				A	Value1
Prod04				B	Value2
Prod05	eeee				
Prod06	ffff				

Table 6 - Theoretical Example for landing.PriceRules

Looking at the landing.PriceRules table, the Customer Code is missing for the 3rd and 4th records. For these records we need to analyze the Customer Price Group and Customer Category Code.



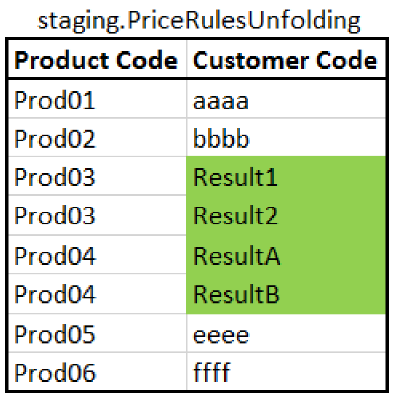
staging.Customer

Customer Code	A	B
Result1	Value1	
Result2	Value1	
ResultA		Value2
ResultB		Value2

Table 7 - Theoretical Example for staging.Customer

The Customer Price Group refers to the column used to lookup on the staging.Customer table, which in this case is column “A”. The Customer Category Code refers to the value used to lookup in the column A, which is “Value1”. On the staging.Customer table, by checking the column A with the Value1, there are 2 Customer Codes, “Result1” and “Result2”. This means that the 3rd record of the APA will unfold into 2 records, as can be verified in the staging.PriceRulesUnfolding table. The same logic is applied to the 4th record of the landing.PriceRules table and again it unfolds to the Customer Codes that were found in Column B with the value “Value2” in the staging.Customer table, which are “ResultA” and “ResultB”.

The final result has a combination of Product Code and Customer Code for every Advanced Pricing Agreement that exists in the landing.PriceRules table.



staging.PriceRulesUnfolding

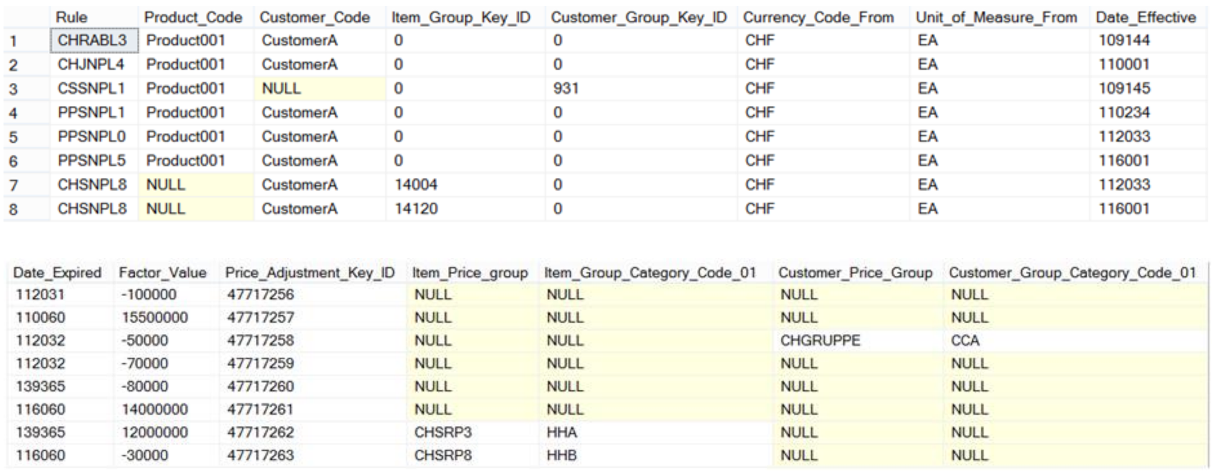
Product Code	Customer Code
Prod01	aaaa
Prod02	bbbb
Prod03	Result1
Prod03	Result2
Prod04	ResultA
Prod04	ResultB
Prod05	eeee
Prod06	ffff

Table 8 - Theoretical Example for staging.PriceRulesUnfolding

Project Case

At the project level, the logic is the same, but with more details to work. The objective remains to unfold all the groups to get the product code and customer code combinations for every advanced pricing agreement.

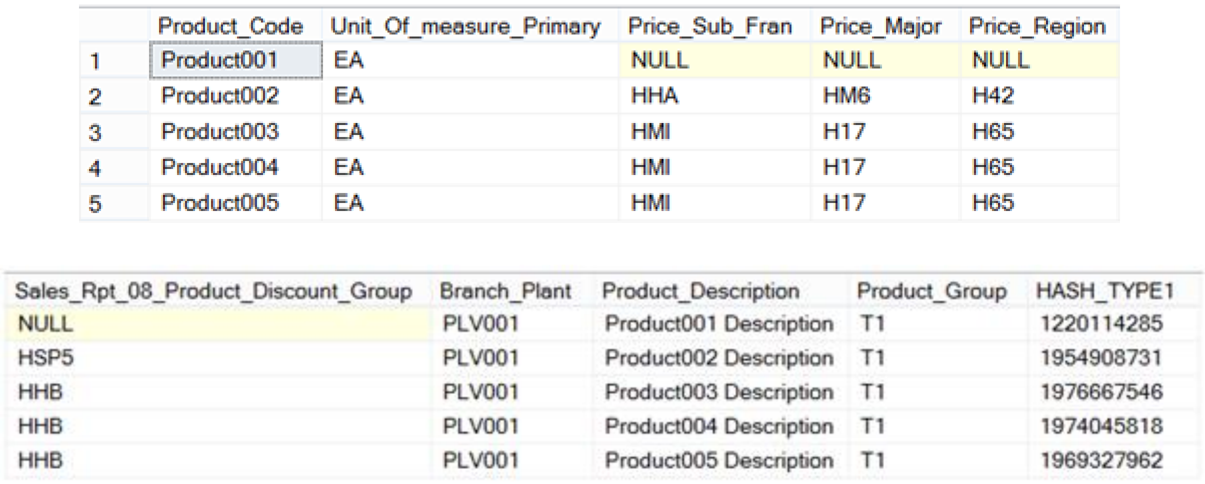
As explained before, on the landing.PriceRules table the agreement is set at the product code or product group level and at the customer code or customer group level. Therefore, each time the code is empty it means that the agreement is set at the group level, which is comprised of multiple codes.



	Rule	Product_Code	Customer_Code	Item_Group_Key_ID	Customer_Group_Key_ID	Currency_Code_From	Unit_of_Measure_From	Date_Effective	Date_Expired	Factor_Value	Price_Adjustment_Key_ID	Item_Price_group	Item_Group_Category_Code_01	Customer_Price_Group	Customer_Group_Category_Code_01
1	CHRABL3	Product001	CustomerA	0	0	CHF	EA	109144	112031	-100000	47717256	NULL	NULL	NULL	NULL
2	CHJNPL4	Product001	CustomerA	0	0	CHF	EA	110001	110060	15500000	47717257	NULL	NULL	NULL	NULL
3	CSSNPL1	Product001	NULL	0	931	CHF	EA	109145	112032	-50000	47717258	NULL	NULL	CHGRUPPE	CCA
4	PPSNPL1	Product001	CustomerA	0	0	CHF	EA	110234	112032	-70000	47717259	NULL	NULL	NULL	NULL
5	PPSNPL0	Product001	CustomerA	0	0	CHF	EA	112033	139365	-80000	47717260	NULL	NULL	NULL	NULL
6	PPSNPL5	Product001	CustomerA	0	0	CHF	EA	116001	116060	14000000	47717261	NULL	NULL	NULL	NULL
7	CHSNPL8	NULL	CustomerA	14004	0	CHF	EA	112033	139365	12000000	47717262	CHSRP3	HHA	NULL	NULL
8	CHSNPL8	NULL	CustomerA	14120	0	CHF	EA	116001	116060	-30000	47717263	CHSRP8	HHB	NULL	NULL

Table 9 - Example records from landing.PriceRules

Whenever a Product_Code is blank on the landing.PriceRules table, the Item_Price_Group and Item_Group_Category_Code_01 columns represent the product group level and hold the necessary information to lookup the product codes from the staging.Product. The Item_Price_Group column holds the data that points to the column to lookup on the staging.Product table. The Item_Group_Category_Code_01 column holds the data to search for in the lookup column determined by the Item_Price_Group.



	Product_Code	Unit_Of_measure_Primary	Price_Sub_Fran	Price_Major	Price_Region	Sales_Rpt_08_Product_Discount_Group	Branch_Plant	Product_Description	Product_Group	HASH_TYPE1
1	Product001	EA	NULL	NULL	NULL	NULL	PLV001	Product001 Description	T1	1220114285
2	Product002	EA	HHA	HM6	H42	HSP5	PLV001	Product002 Description	T1	1954908731
3	Product003	EA	HMI	H17	H65	HHB	PLV001	Product003 Description	T1	1976667546
4	Product004	EA	HMI	H17	H65	HHB	PLV001	Product004 Description	T1	1974045818
5	Product005	EA	HMI	H17	H65	HHB	PLV001	Product005 Description	T1	1969327962

Table 10 - Example records from staging.Product

The column Item_Price_Group contains 4 codes that point to 4 columns in the staging.Product table:

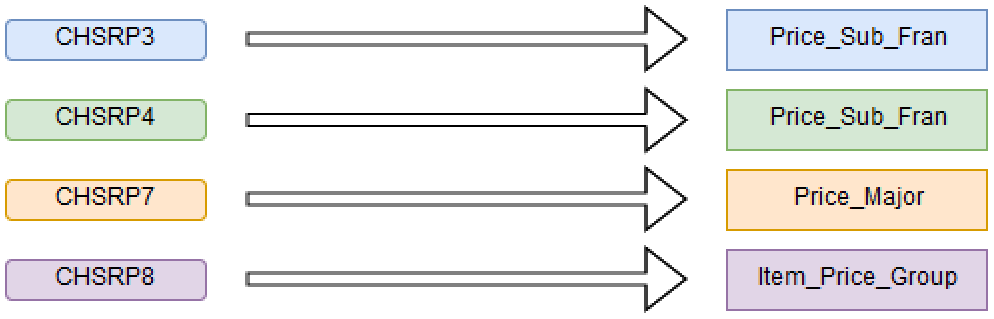


Figure 13 - Item Price Group Logic

The same relationship can be verified for the customer group level, which is also comprised of two columns. The column Customer_Price_Group holds the codes to lookup the column on the staging.Customer table, which in this case is always one code (CHGRUPPE), and the column Customer_Group_Category_Code_01 holds the data to search for in that lookup column (Customer_Group).

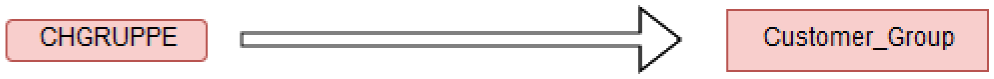
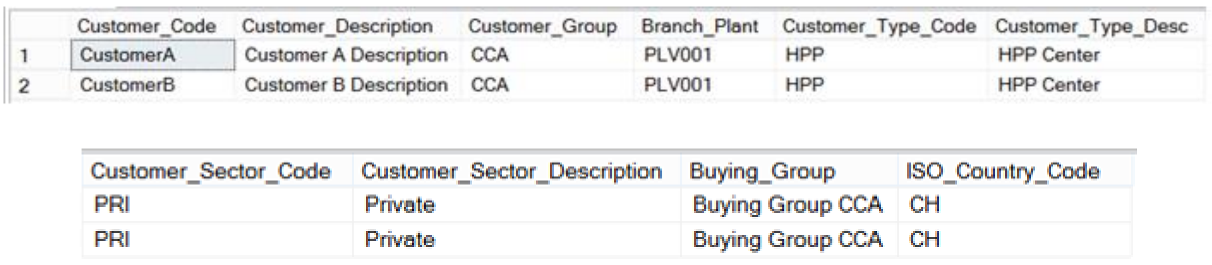


Figure 14 - Customer Price Group Logic



	Customer_Code	Customer_Description	Customer_Group	Branch_Plant	Customer_Type_Code	Customer_Type_Desc
1	CustomerA	Customer A Description	CCA	PLV001	HPP	HPP Center
2	CustomerB	Customer B Description	CCA	PLV001	HPP	HPP Center

Customer_Sector_Code	Customer_Sector_Description	Buying_Group	ISO_Country_Code
PRI	Private	Buying Group CCA	CH
PRI	Private	Buying Group CCA	CH

Table 11 - Example records from staging.Customer

Putting everything together, the unfolding process works as follows:

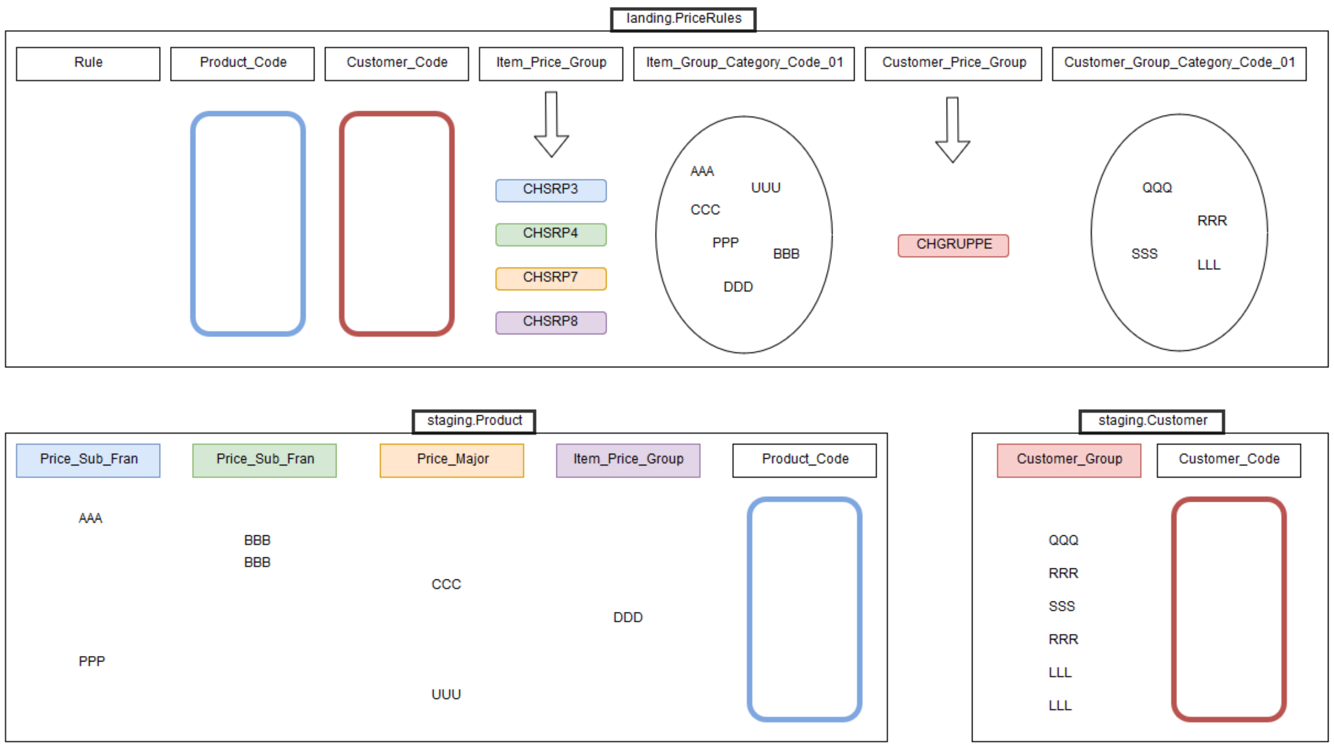
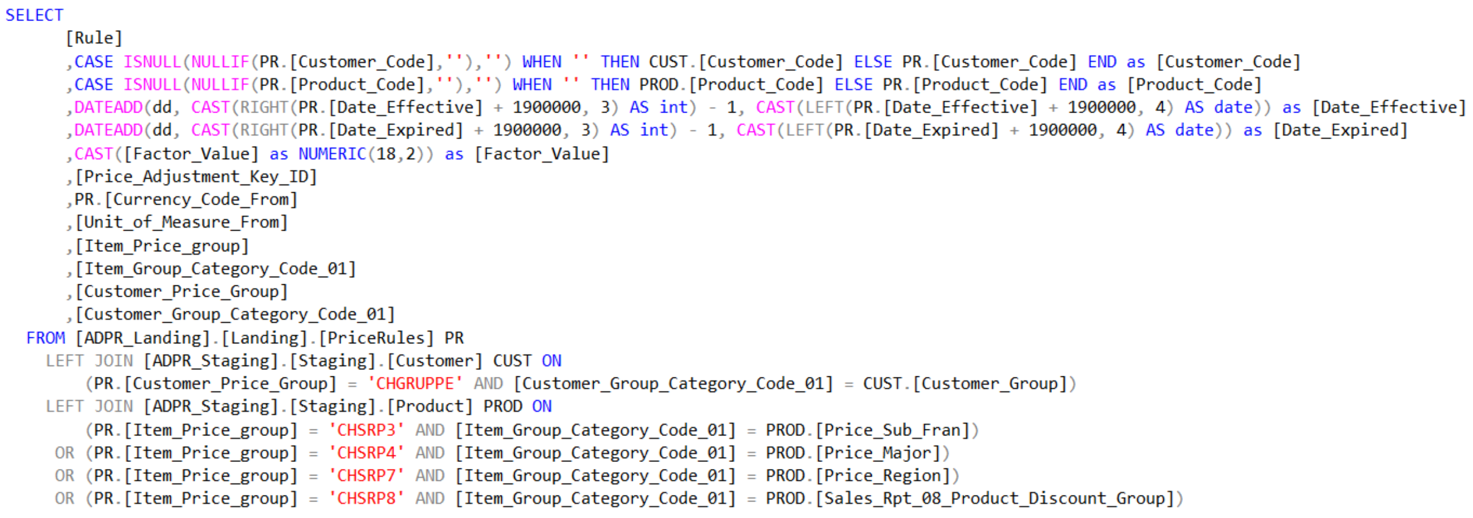


Figure 15 - Unfolding Process Model

The 3 tables are represented individually. The product codes and customer codes can already come in blue and red areas in the landing table or in case they are blank, they will be looked up in the staging tables staging.Product and staging.Customer through the unfolding logic by using the data on their respective product group and customer group columns.

Query used to achieve the unfolding logic in the project:



```sql
SELECT
      [Rule]
      ,CASE ISNULL(NULLIF(PR.[Customer_Code],''),'') WHEN '' THEN CUST.[Customer_Code] ELSE PR.[Customer_Code] END as [Customer_Code]
      ,CASE ISNULL(NULLIF(PR.[Product_Code],''),'') WHEN '' THEN PROD.[Product_Code] ELSE PR.[Product_Code] END as [Product_Code]
      ,DATEADD(dd, CAST(RIGHT(PR.[Date_Effective] + 1900000, 3) AS int) - 1, CAST(LEFT(PR.[Date_Effective] + 1900000, 4) AS date)) as [Date_Effective]
      ,DATEADD(dd, CAST(RIGHT(PR.[Date_Expired] + 1900000, 3) AS int) - 1, CAST(LEFT(PR.[Date_Expired] + 1900000, 4) AS date)) as [Date_Expired]
      ,CAST([Factor_Value] as NUMERIC(18,2)) as [Factor_Value]
      ,[Price_Adjustment_Key_ID]
      ,PR.[Currency_Code_From]
      ,[Unit_of_Measure_From]
      ,[Item_Price_group]
      ,[Item_Group_Category_Code_01]
      ,[Customer_Price_Group]
      ,[Customer_Group_Category_Code_01]
  FROM [ADPR_Landing].[Landing].[PriceRules] PR
    LEFT JOIN [ADPR_Staging].[Staging].[Customer] CUST ON
        (PR.[Customer_Price_Group] = 'CHGRUPPE' AND [Customer_Group_Category_Code_01] = CUST.[Customer_Group])
    LEFT JOIN [ADPR_Staging].[Staging].[Product] PROD ON
        (PR.[Item_Price_group] = 'CHSRP3' AND [Item_Group_Category_Code_01] = PROD.[Price_Sub_Fran])
     OR (PR.[Item_Price_group] = 'CHSRP4' AND [Item_Group_Category_Code_01] = PROD.[Price_Major])
     OR (PR.[Item_Price_group] = 'CHSRP7' AND [Item_Group_Category_Code_01] = PROD.[Price_Region])
     OR (PR.[Item_Price_group] = 'CHSRP8' AND [Item_Group_Category_Code_01] = PROD.[Sales_Rpt_08_Product_Discount_Group])
```

Figure 16 - Query Unfolding

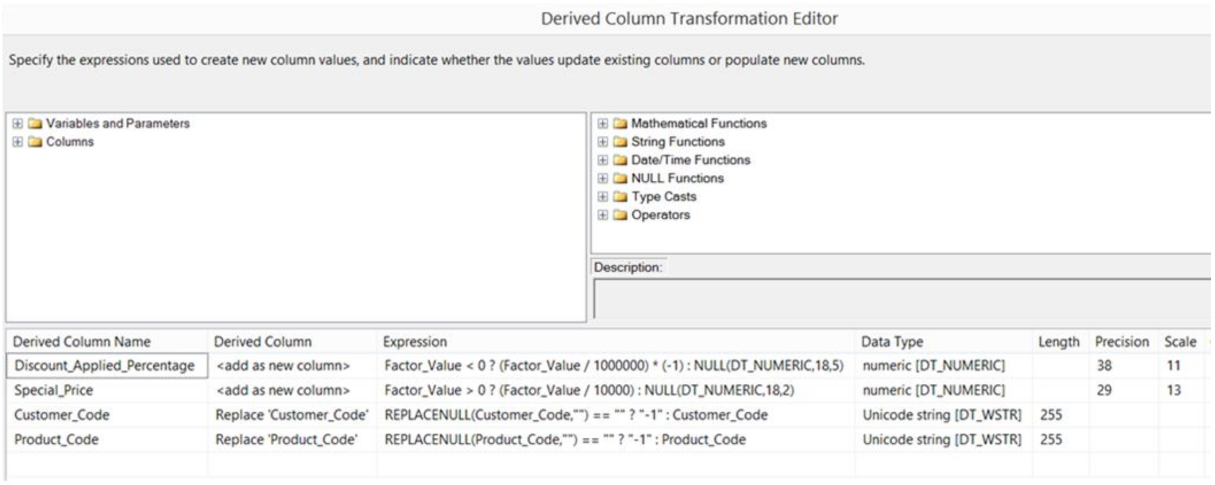
The unfolding logic in the query is applied in the column [Customer_Code], [Product_Code] and the LEFT JOINs.

The [Date_Effective] and [Date_Expired] columns were in JDE Julian Date format and were converted to Gregorian Date. Explanation and validations can be found in the attachments in subchapter 7.1.

Whenever an agreement has no Product Code and no Customer Code, it means the agreement is applied at their group levels, meaning that a Cartesian product takes effect to obtain all the possible combinations of Product Code and Customer Code. The query also upholds this. Validations are provided in the attachments subchapter 7.2.

The mappings are hardcoded in the LEFT JOINS and no mapping configurable table was created because it is unjustified. The business analyst assured that only the 4 Item_Price_Groups and the Customer_Price_Group presented above were needed and no plans existed to include more. Furthermore, a new Item_Price_Group or Customer_Price_Group would mean a new column in the Product or Customer tables, thus requiring new developments.

Additional transformations were made using a Derived Column Transformation Editor shown in Figure 17:


Derived Column Transformation Editor
Specify the expressions used to create new column values, and indicate whether the values update existing columns or populate new columns.
Variables and Parameters
Columns
Mathematical Functions
String Functions
Date/Time Functions
NULL Functions
Type Casts
Operators
Description:

Derived Column Name	Derived Column	Expression	Data Type	Length	Precision	Scale
Discount_Applied_Percentage	<add as new column>	Factor_Value < 0 ? (Factor_Value / 1000000) * (-1) : NULL(DT_NUMERIC,18,5)	numeric [DT_NUMERIC]		38	11
Special_Price	<add as new column>	Factor_Value > 0 ? (Factor_Value / 10000) : NULL(DT_NUMERIC,18,2)	numeric [DT_NUMERIC]		29	13
Customer_Code	Replace 'Customer_Code'	REPLACENULL(Customer_Code,"") == "" ? "-1" : Customer_Code	Unicode string [DT_WSTR]	255		
Product_Code	Replace 'Product_Code'	REPLACENULL(Product_Code,"") == "" ? "-1" : Product_Code	Unicode string [DT_WSTR]	255		

Figure 17 - Unfolding - SSIS Derived Column

The Factor_Value Column refers to price adjustment details in the JDE System (JD Edwards, 2020) and can have positive and negative values. The negative values indicate discounts in percentage and positive values refer to special fixed prices that override the list price of a particular product. For the negative values, using the expression in the Figure 17, the value is divided by 1.000.000 and multiplied by -1 in order to properly convert these JDE values into standard percentages.

For the positive values, the expression is a little different, as the value needs to be divided by 10.000 to convert it to the special price.

The Customer_Code and Product_Code, whenever NULL, were replaced by the inferred member -1.

At the end, the package responsible for executing the transformations of the Unfolding logic is executed with success.

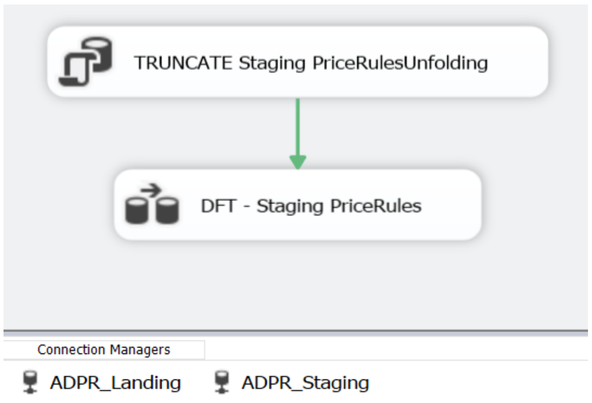


Figure 18 - SSIS Control Flow - Unfolding

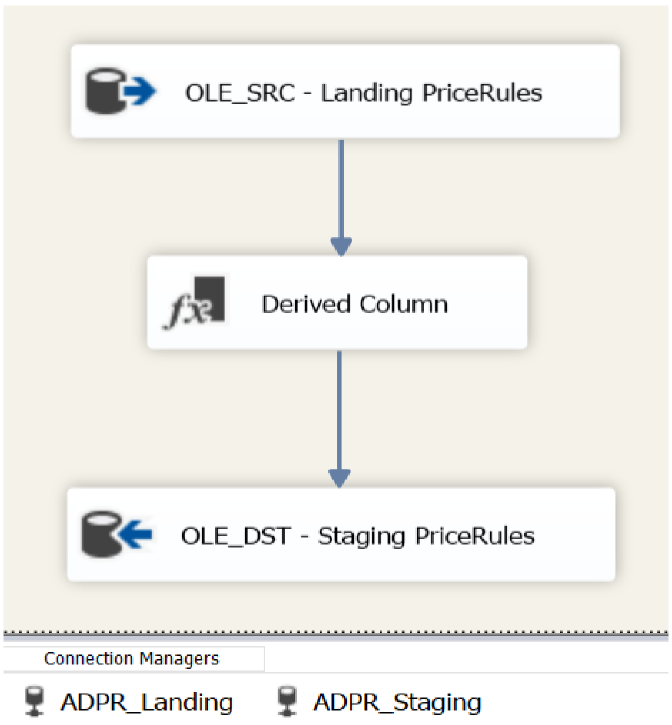
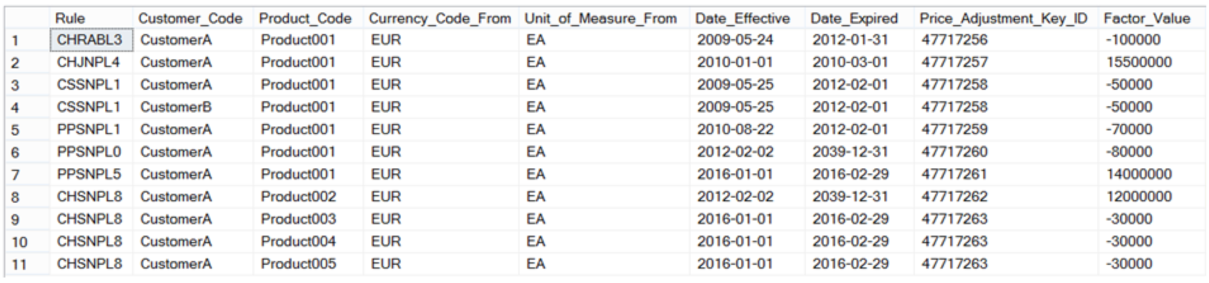
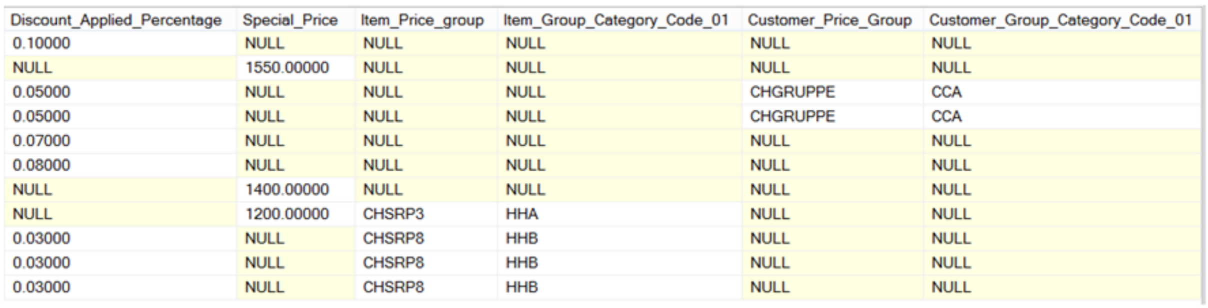


Figure 19 - SSIS Data Flow - Unfolding

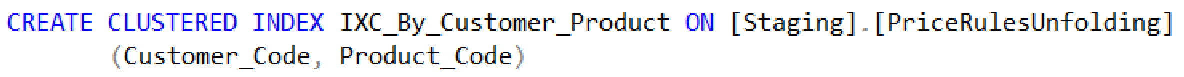
The data is inserted in the table staging.PriceRulesUnfolding as such:

	Rule	Customer_Code	Product_Code	Currency_Code_From	Unit_of_Measure_From	Date_Effective	Date_Expired	Price_Adjustment_Key_ID	Factor_Value	Discount_Applied_Percentage	Special_Price	Item_Price_group	Item_Group_Category_Code_01	Customer_Price_Group	Customer_Group_Category_Code_01
1	CHRABL3	CustomerA	Product001	EUR	EA	2009-05-24	2012-01-31	47717256	-100000	0.10000	NULL	NULL	NULL	NULL	NULL
2	CHJNPL4	CustomerA	Product001	EUR	EA	2010-01-01	2010-03-01	47717257	15500000	NULL	1550.00000	NULL	NULL	NULL	NULL
3	CSSNPL1	CustomerA	Product001	EUR	EA	2009-05-25	2012-02-01	47717258	-50000	0.05000	NULL	NULL	NULL	CHGRUPPE	CCA
4	CSSNPL1	CustomerB	Product001	EUR	EA	2009-05-25	2012-02-01	47717258	-50000	0.05000	NULL	NULL	NULL	CHGRUPPE	CCA
5	PPSNPL1	CustomerA	Product001	EUR	EA	2010-08-22	2012-02-01	47717259	-70000	0.07000	NULL	NULL	NULL	NULL	NULL
6	PPSNPL0	CustomerA	Product001	EUR	EA	2012-02-02	2039-12-31	47717260	-80000	0.08000	NULL	NULL	NULL	NULL	NULL
7	PPSNPL5	CustomerA	Product001	EUR	EA	2016-01-01	2016-02-29	47717261	14000000	NULL	1400.00000	NULL	NULL	NULL	NULL
8	CHSNPL8	CustomerA	Product002	EUR	EA	2012-02-02	2039-12-31	47717262	12000000	NULL	1200.00000	CHSRP3	HHA	NULL	NULL
9	CHSNPL8	CustomerA	Product003	EUR	EA	2016-01-01	2016-02-29	47717263	-30000	0.03000	NULL	CHSRP8	HHB	NULL	NULL
10	CHSNPL8	CustomerA	Product004	EUR	EA	2016-01-01	2016-02-29	47717263	-30000	0.03000	NULL	CHSRP8	HHB	NULL	NULL
11	CHSNPL8	CustomerA	Product005	EUR	EA	2016-01-01	2016-02-29	47717263	-30000	0.03000	NULL	CHSRP8	HHB	NULL	NULL

Table 12 - Unfolding - Final Result

A clustered rowstore index was created in order to store the table data by sorting it for Customer_Code and Product_Code to facilitate the table scan for the next phase of the transformation – Date Ranges and List Price.



```sql
CREATE CLUSTERED INDEX IXC_By_Customer_Product ON [Staging].[PriceRulesUnfolding]
       (Customer_Code, Product_Code)
```

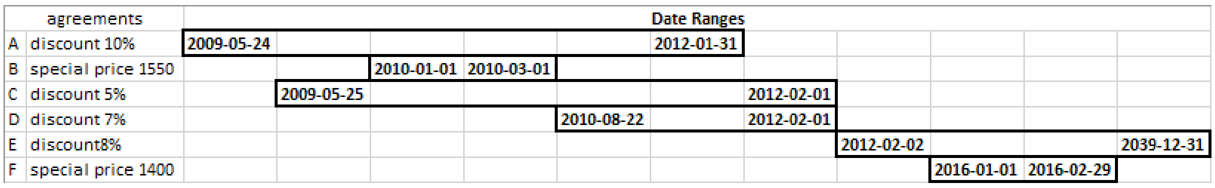
Figure 20 - Clustered Rowstore Index for staging.PriceRulesUnfolding

3.3.2.2. Date Ranges and List Price

After having all the discounts presented at the customer code and product code levels in the unfolding phase, the next step is to address the overlap of the discounts and appropriately divide the date ranges. Since multiple discounts can be active in the same period, it's necessary to group them into common date ranges in order to have a better understanding of all the agreements that are in place.

Theoretical Example

To demonstrate, the following example shows 6 different advanced pricing agreements (A, B, C, D, E and F) in different time periods for the same customer and product, which are the same CustomerA and Product001 from the previous example. The dates in **bold** are the original Date Effective and Date Expired for each discount.

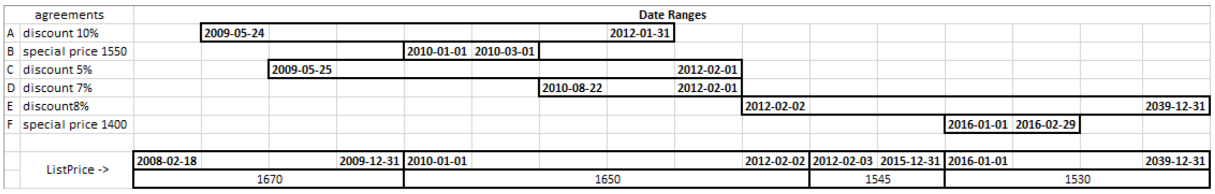


	agreements	Date Ranges										
A	discount 10%	2009-05-24					2012-01-31					
B	special price 1550			2010-01-01	2010-03-01							
C	discount 5%		2009-05-25					2012-02-01				
D	discount 7%					2010-08-22		2012-02-01				
E	discount8%								2012-02-02			2039-12-31
F	special price 1400									2016-01-01	2016-02-29	

Figure 21 - Date Ranges for CustomerA and Product001

Since each discount involves a product, that product must have a list price. The list price is always in effect and acts as the final price whenever there are no discounts and as base price to calculate the final price after the discounts have been applied.

Adding the list price to the example, the situation becomes more complex. The list price must always exist and it also has a Date Effective and Date Expired for every price it holds. The value of the list price is presented below the respective date ranges in Figure 22.

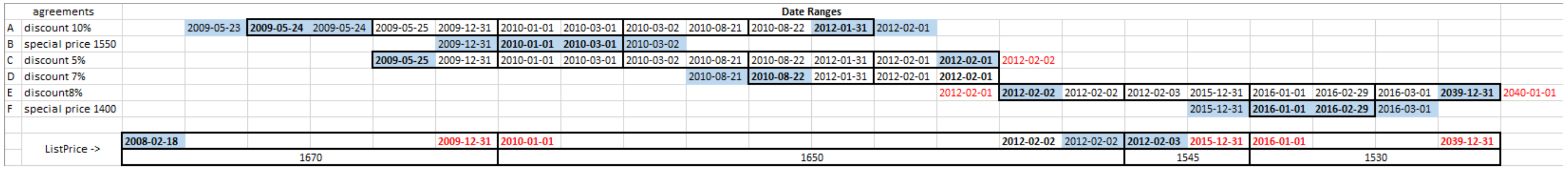


	agreements	Date Ranges															
A	discount 10%		2009-05-24						2012-01-31								
B	special price 1550					2010-01-01	2010-03-01										
C	discount 5%			2009-05-25						2012-02-01							
D	discount 7%							2010-08-22		2012-02-01							
E	discount8%										2012-02-02						2039-12-31
F	special price 1400													2016-01-01	2016-02-29		
	ListPrice ->	2008-02-18			2009-12-31	2010-01-01					2012-02-02	2012-02-03	2015-12-31	2016-01-01			2039-12-31
				1670					1650			1545				1530	

Figure 22 - Date Ranges and List price for CustomerA and Product001

The objective is to have a date effective and date expired for any combination of discounts that are being applied at that same date range.

By correctly grouping these dates and applying the date ranges aggregation rules that will be presented in Table 14 in the Project Case, we get the following result illustrated in Figure 23:

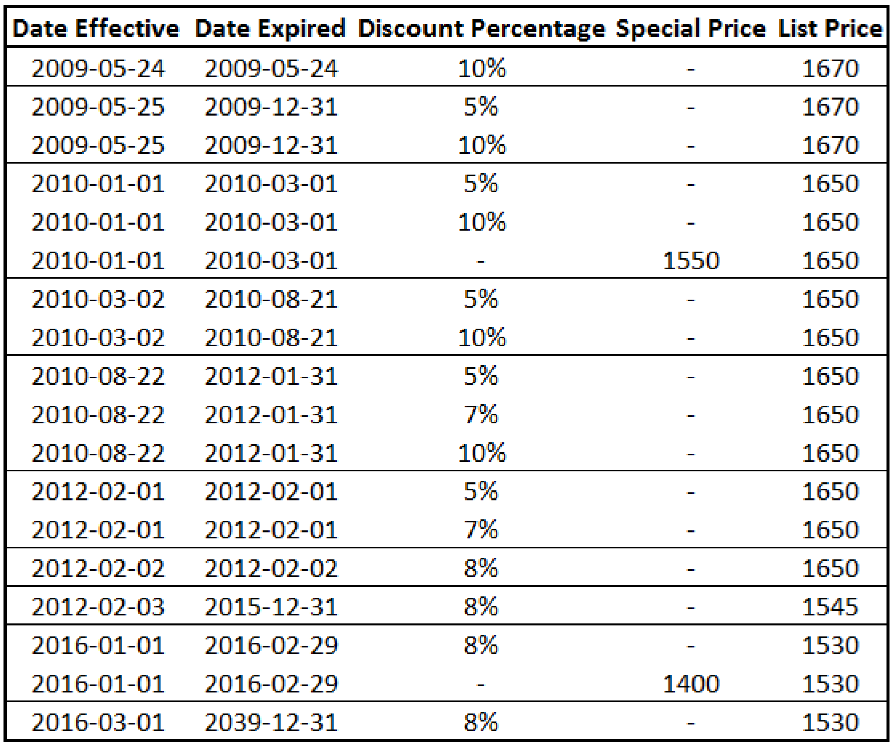


	agreements	Date Ranges																					
A	discount 10%		2009-05-23	2009-05-24	2009-05-24	2009-05-25	2009-12-31	2010-01-01	2010-03-01	2010-03-02	2010-08-21	2010-08-22	2012-01-31	2012-02-01									
B	special price 1550						2009-12-31	2010-01-01	2010-03-01	2010-03-02													
C	discount 5%					2009-05-25	2009-12-31	2010-01-01	2010-03-01	2010-03-02	2010-08-21	2010-08-22	2012-01-31	2012-02-01	2012-02-01	2012-02-02							
D	discount 7%										2010-08-21	2010-08-22	2012-01-31	2012-02-01	2012-02-01								
E	discount8%														2012-02-01	2012-02-02	2012-02-02	2012-02-03	2015-12-31	2016-01-01	2016-02-29	2016-03-01	2039-12-31	2040-01-01
F	special price 1400																		2015-12-31	2016-01-01	2016-02-29	2016-03-01		
	ListPrice ->	2008-02-18					2009-12-31	2010-01-01								2012-02-02	2012-02-02	2012-02-03	2015-12-31	2016-01-01			2039-12-31	
					1670								1650						1545			1530		

Figure 23 - Date Ranges and list Price segregated for CustomerA and Product001

The blue dates indicate the dates that are used to correctly divide the discounts into common date ranges. The red dates were not inserted because they did not meet the conditions of the rules. They are shown in red to exemplify the cases that would otherwise be used to make the divisions.

The result of this transformation that needs to occur in order to correctly group the discounts into common date ranges is presented in the Table 13:



Date Effective	Date Expired	Discount Percentage	Special Price	List Price
2009-05-24	2009-05-24	10%	-	1670
2009-05-25	2009-12-31	5%	-	1670
2009-05-25	2009-12-31	10%	-	1670
2010-01-01	2010-03-01	5%	-	1650
2010-01-01	2010-03-01	10%	-	1650
2010-01-01	2010-03-01	-	1550	1650
2010-03-02	2010-08-21	5%	-	1650
2010-03-02	2010-08-21	10%	-	1650
2010-08-22	2012-01-31	5%	-	1650
2010-08-22	2012-01-31	7%	-	1650
2010-08-22	2012-01-31	10%	-	1650
2012-02-01	2012-02-01	5%	-	1650
2012-02-01	2012-02-01	7%	-	1650
2012-02-02	2012-02-02	8%	-	1650
2012-02-03	2015-12-31	8%	-	1545
2016-01-01	2016-02-29	8%	-	1530
2016-01-01	2016-02-29	-	1400	1530
2016-03-01	2039-12-31	8%	-	1530

Table 13 - Final expected result for CustomerA and Product001

With the date ranges now grouped and arranged, it is clear which discounts are valid for any specific day. This table displays the agreements from Figure 23 in an organized tabular structure for better understanding. This is also the result we are looking for in the ETL process of this phase.

For example, in any day within the 2010-01-01 and 2010-03-31 date range, if a purchase is made by this customer and product, then agreement A (10%), B (1550) and C (5%) are effective and will be cast upon that product’s list price to calculate de final price.

Project Case

For better understanding, the theoretical example was used as the input data to illustrate the technical transformations that allow multiple date ranges that need to be correctly segmented in a single combination of product and customer.

To do this, a set of rules were created in order to correctly divide the discounts while taking into account the associated list prices. Table 14 is the result of testing all the possible scenarios in order to define all the rules that will correctly separate any advanced pricing scenario into common date ranges to give visibility of every different discount taking place no matter the complexity behind it. All the testing to demonstrate the table below can be found in the attachments in Subchapter 7.4).

Aggregation rules to correctly divide the discounts and their List Prices into common date ranges:

	Insert Distinct Dates	**Conditions**
Discounts	Date Effective (-1)	when there is no Date Expired
	Date Effective	always
	Date Expired	always
	Date Expired (+1)	when there is no Date Effective
ListPrice	Date Effective	when there is Discount Date Effective(-1) when there is no Discount Date Effective when there is no Discount Date Expired(+1)
		when there is Discount Date Expired when there is no Discount Date Effective when there is no Discount Date Expired(+1)
		when there is no Discount Date Effective(-1) when there is no Discount Date Effective when there is no Discount Date Expired when there is no Discount Date Expired(+1)
	Date Expired	when there is Discount Date Expired(+1) when there is no Discount Date Expired when there is no Discount Date Effective(-1)
		when there is Discount Date Effective when there is no Discount Date Expired when there is no Discount Date Effective(-1)
		when there is no Discount Date Effective(-1) when there is no Discount Date Effective when there is no Discount Date Expired when there is no Discount Date Expired(+1)

Table 14 - Date Ranges Aggregation Rules

The aggregation rules are applied considering only the date ranges of the discounts and list prices of a specific customer code and product code at a time. Whenever these rules are applied, any other existing dates from discounts and list prices from another combination of customer code and product code are completely excluded from the process taking place.

Using Figure 23 as the illustration reference, the first phase to create the date ranges aggregation rules is to define the foundation dates and the outer dates. These are comprised of the distinct Date Effective and Date Expired of all the discounts for the same customer and product. Their outer foundations consist of a day before the Date Effective and a day after the Date Expired.

Example: Discount valid from 2010-01-01 to 2010-03-01

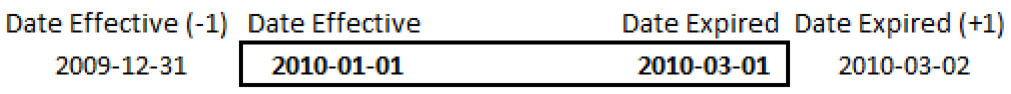


Figure 24 - Discount Example

For this discount, the foundations are the following:

Date Effective: 2010-01-01
Date Expired: 2010-03-01

And the outer dates are:

Date Effective (-1): 2009-12-31
Date Expired (+1): 2010-03-02

This process is done to every discount for that particular customer and product.

The Date Effective and Date Expired of all the discounts are always considered once, meaning that if 3 discounts have the same Date Effective, this date will be used once.

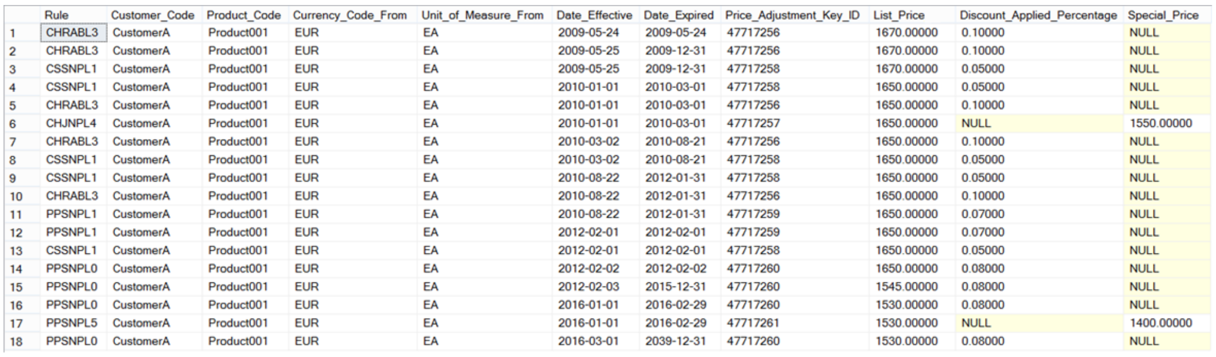
The Date Effective (-1) is only applied when there is no Date Expired from another discount overlapping that same date. Now in reverse, the Date Expired (+1) is only applied when there is no Date Effective from another discount on the same date.

The second phase is to include the List Price and decide when its Date Effective and Date Expired will be considered as a date. The second phase rules already consider that the first phase rules are in place. Furthermore, the conditions of the second phase all refer to the discount dates. Hence, looking at the first example of Table 14 of the List Price, the Date Effective will be considered in case there is a Date Effective (-1) that exists from at least one discount for the same customer code and product code and there is no Date Effective or Date Expired (+1) from any discounts on that same date.

Looking at another example, the Date Expired of the List Price will always be considered if there is no foundation and outer dates from any discount of the same customer code and product code. This also applies for the Date Effective of the List Price, which will always be considered in case there is no Date Effective (-1), Date Effective, Date Expired or Date Expired (+1) of any discount for that same date.

The query used to separate the dates into common date ranges for all the combinations of customers and products and all the details regarding the logic behind it can be found in the attachments in subchapter 7.3.1.

In the end the query will generate the following results for the example case presented:



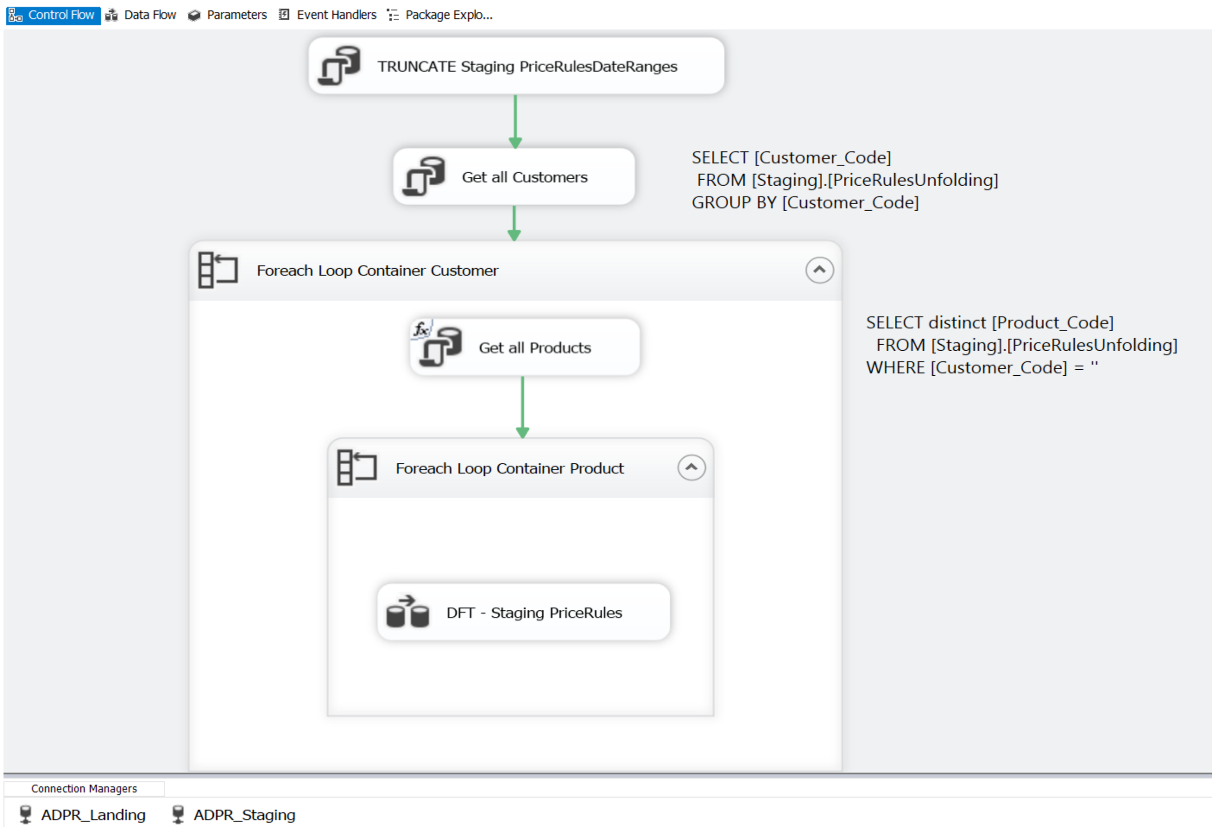
	Rule	Customer_Code	Product_Code	Currency_Code_From	Unit_of_Measure_From	Date_Effective	Date_Expired	Price_Adjustment_Key_ID	List_Price	Discount_Applied_Percentage	Special_Price
1	CHRABL3	CustomerA	Product001	EUR	EA	2009-05-24	2009-05-24	47717256	1670.00000	0.10000	NULL
2	CHRABL3	CustomerA	Product001	EUR	EA	2009-05-25	2009-12-31	47717256	1670.00000	0.10000	NULL
3	CSSNPL1	CustomerA	Product001	EUR	EA	2009-05-25	2009-12-31	47717258	1670.00000	0.05000	NULL
4	CSSNPL1	CustomerA	Product001	EUR	EA	2010-01-01	2010-03-01	47717258	1650.00000	0.05000	NULL
5	CHRABL3	CustomerA	Product001	EUR	EA	2010-01-01	2010-03-01	47717256	1650.00000	0.10000	NULL
6	CHJNPL4	CustomerA	Product001	EUR	EA	2010-01-01	2010-03-01	47717257	1650.00000	NULL	1550.00000
7	CHRABL3	CustomerA	Product001	EUR	EA	2010-03-02	2010-08-21	47717256	1650.00000	0.10000	NULL
8	CSSNPL1	CustomerA	Product001	EUR	EA	2010-03-02	2010-08-21	47717258	1650.00000	0.05000	NULL
9	CSSNPL1	CustomerA	Product001	EUR	EA	2010-08-22	2012-01-31	47717258	1650.00000	0.05000	NULL
10	CHRABL3	CustomerA	Product001	EUR	EA	2010-08-22	2012-01-31	47717256	1650.00000	0.10000	NULL
11	PPSNPL1	CustomerA	Product001	EUR	EA	2010-08-22	2012-01-31	47717259	1650.00000	0.07000	NULL
12	PPSNPL1	CustomerA	Product001	EUR	EA	2012-02-01	2012-02-01	47717259	1650.00000	0.07000	NULL
13	CSSNPL1	CustomerA	Product001	EUR	EA	2012-02-01	2012-02-01	47717258	1650.00000	0.05000	NULL
14	PPSNPL0	CustomerA	Product001	EUR	EA	2012-02-02	2012-02-02	47717260	1650.00000	0.08000	NULL
15	PPSNPL0	CustomerA	Product001	EUR	EA	2012-02-03	2015-12-31	47717260	1545.00000	0.08000	NULL
16	PPSNPL0	CustomerA	Product001	EUR	EA	2016-01-01	2016-02-29	47717260	1530.00000	0.08000	NULL
17	PPSNPL5	CustomerA	Product001	EUR	EA	2016-01-01	2016-02-29	47717261	1530.00000	NULL	1400.00000
18	PPSNPL0	CustomerA	Product001	EUR	EA	2016-03-01	2039-12-31	47717260	1530.00000	0.08000	NULL

Table 15 - Date Ranges - Full Query Results

The query results in Table 15 match exactly the expected results from manually breaking down the discounts into common date ranges, shown in Figure 23.

SSIS Control Flow

The SSIS Control Flow is responsible for the process of taking every combination of customer and product from the staging.PriceRulesUnfolding and separating their discounts into common date ranges is shown in Figure 25:



```
SELECT [Customer_Code]
 FROM [Staging].[PriceRulesUnfolding]
GROUP BY [Customer_Code]
```

```
SELECT distinct [Product_Code]
  FROM [Staging].[PriceRulesUnfolding]
WHERE [Customer_Code] = ''
```

Figure 25 - Date Ranges - SSIS Control Flow

In order to cycle every combination of customer and product, one at a time to prevent other dates from interfering, a nested foreach loop container was implemented. All details regarding this process are found in the subchapter 7.3.2.

SSIS Data Flow

The last component in the Control Flow is the Data Flow Task, which contains only 2 components: an OLE DB Source and a Destination component. The source component is where the actual query from Figure 52 is processed using the SQL command from the variable QUERY, as shown in Figure 27.

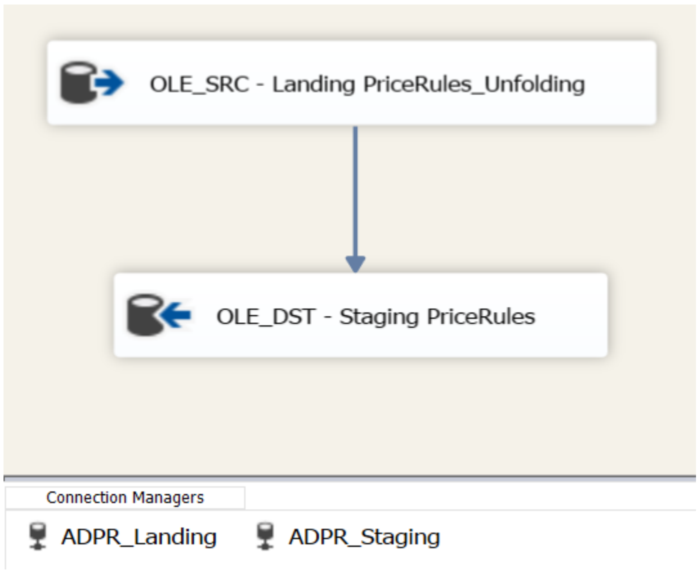
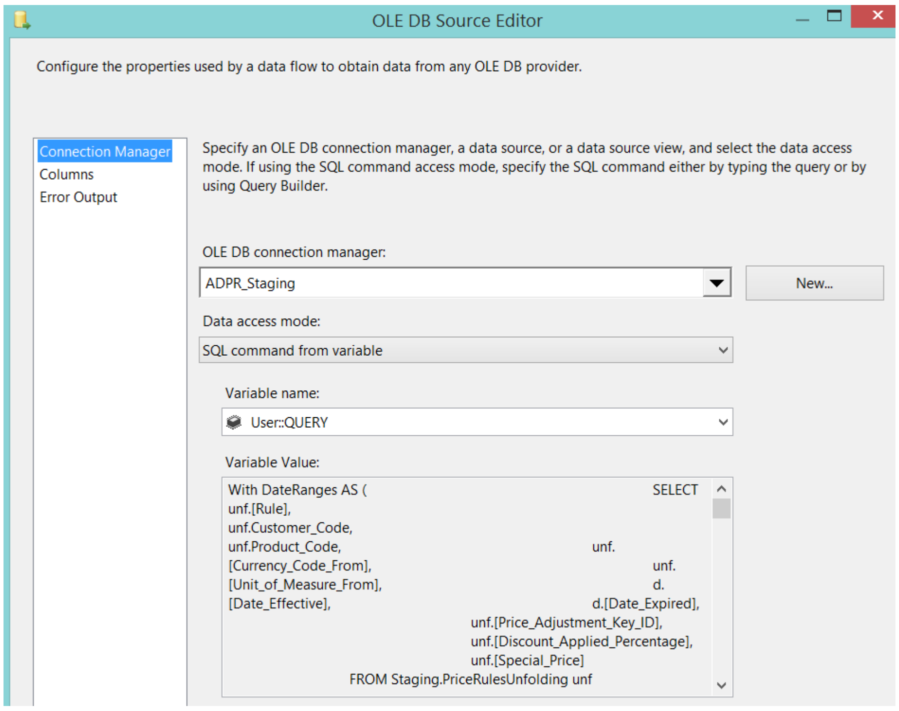

OLE_SRC - Landing PriceRules_Unfolding
OLE_DST - Staging PriceRules
Connection Managers
ADPR_Landing
ADPR_Staging

Figure 26 - Date Ranges - Data Flow



Configure the properties used by a data flow to obtain data from any OLE DB provider.

Specify an OLE DB connection manager, a data source, or a data source view, and select the data access mode. If using the SQL command access mode, specify the SQL command either by typing the query or by using Query Builder.

OLE DB connection manager: ADPR_Staging

Data access mode: SQL command from variable

Variable name: User::QUERY

Variable Value:

```
With DateRanges AS (                                         SELECT
unf.[Rule],
unf.Customer_Code,
unf.Product_Code,                                    unf.
[Currency_Code_From],                                        unf.
[Unit_of_Measure_From],                                      d.
[Date_Effective],                                    d.[Date_Expired],
                         unf.[Price_Adjustment_Key_ID],
                         unf.[Discount_Applied_Percentage],
                         unf.[Special_Price]
            FROM Staging.PriceRulesUnfolding unf
```

Figure 27 - Date Ranges - Source Component

The destination component maps the inserted data into the Staging.PriceRulesDateRanges table.

3.3.2.3. Pivot

Simplified Theoretical Example

The last transformation phase for the advanced pricing agreements is to have every combination of customer and product with a common date range to be represented in a single record, thus having each discount attributed to its own column.

Taking the simplified theoretical example below, the PriceRulesDateRanges table shows 2 combinations of customer and product, in green and blue, each with only one common date range.

PriceRulesDateRanges				
Customer Code	**Product Code**	**Date Effective**	**Date Expired**	**Discount**
CustAAA	Prod001	01-01-2016	31-10-2016	0,1
CustAAA	Prod001	01-01-2016	31-10-2016	0,1
CustAAA	Prod001	01-01-2016	31-10-2016	0,15
CustBBB	Prod002	01-11-2016	15-04-2017	0,2
CustBBB	Prod002	01-11-2016	15-04-2017	0,05

Table 16 - PriceRulesDateRanges

The objective is to transform this data into the table PriceRulesPivot. In this format, it's possible to view all the discounts that are in place for a specific time period in a single record.

PriceRulesPivot								
Customer Code	**Product Code**	**Date Effective**	**Date Expired**	**Discount1**	**Discount2**	**Discount3**	**Discount4**	**Discount5**
CustAAA	Prod001	2016-01-01	2016-10-31	0,1	0,1	0,15	N/A	N/A
CustBBB	Prod002	2016-11-01	2017-04-15	0,05	0,2	N/A	N/A	N/A

Table 17 - PriceRulesPivot

Taking the example of the customer code CustAAA and product code Prod001, it's now easy to verify that there are 3 discounts that are applicable from 2016-01-01 and 2016-10-31.

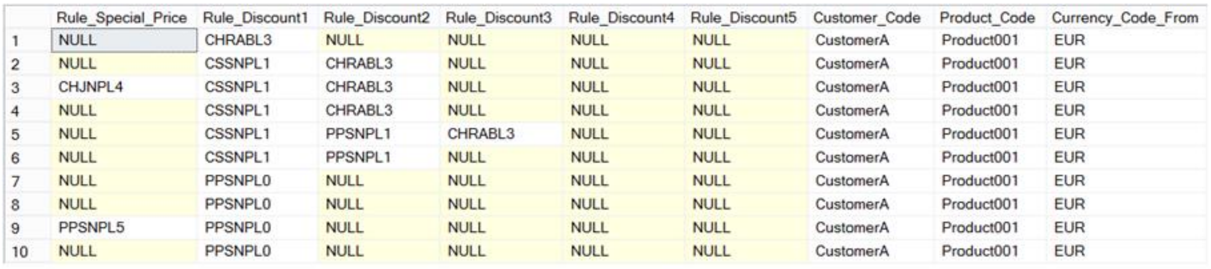
Project Case

Following the same project case example and taking into consideration the query results from Table 15 as the input data for the pivot transformation, the objective is to combine all the special prices and discounts in percentage that occur in the same time period for every customer and product combination into a single record.

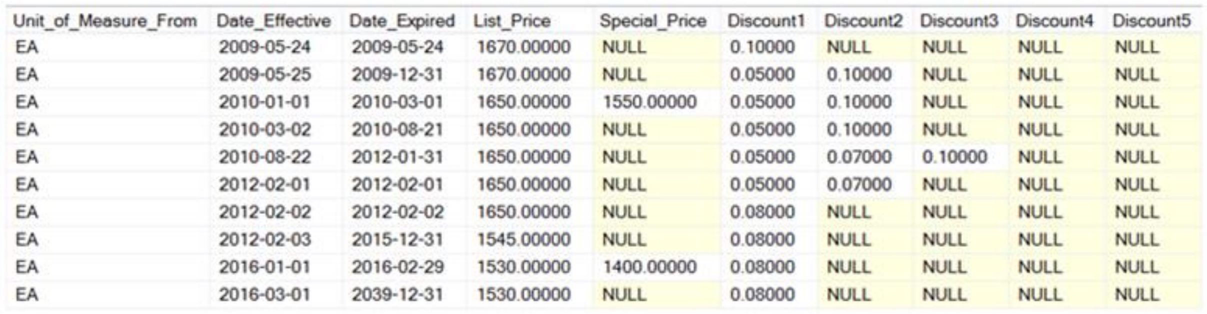
To be able to achieve this level of detail aggregated into a single record, the column Rule had to be extended to 6 columns: one for the special price and five for the possible discounts in percentage occurring at the same time. The same way, the price_adjustment_key_id column had to either be removed or added as 6 additional columns that could either remain in the fact table as degenerate dimensions (Ralph Kimball & Ross, 2013, pp. 37-68) or normalized as a dimension, which would not have further descriptive columns to provide more contexts to the facts. As such, and since there are no near future plans to incorporate the price_adjustment_key_id in front-end presentation purposes, this column was simply removed at this point.

The query used for the Pivot transformation phase along with all the details can be found in the attachments in subchapter 7.5.1.

The query takes the results from the Date Ranges phase and generates the following outcome:



	Rule_Special_Price	Rule_Discount1	Rule_Discount2	Rule_Discount3	Rule_Discount4	Rule_Discount5	Customer_Code	Product_Code	Currency_Code_From
1	NULL	CHRABL3	NULL	NULL	NULL	NULL	CustomerA	Product001	EUR
2	NULL	CSSNPL1	CHRABL3	NULL	NULL	NULL	CustomerA	Product001	EUR
3	CHJNPL4	CSSNPL1	CHRABL3	NULL	NULL	NULL	CustomerA	Product001	EUR
4	NULL	CSSNPL1	CHRABL3	NULL	NULL	NULL	CustomerA	Product001	EUR
5	NULL	CSSNPL1	PPSNPL1	CHRABL3	NULL	NULL	CustomerA	Product001	EUR
6	NULL	CSSNPL1	PPSNPL1	NULL	NULL	NULL	CustomerA	Product001	EUR
7	NULL	PPSNPL0	NULL	NULL	NULL	NULL	CustomerA	Product001	EUR
8	NULL	PPSNPL0	NULL	NULL	NULL	NULL	CustomerA	Product001	EUR
9	PPSNPL5	PPSNPL0	NULL	NULL	NULL	NULL	CustomerA	Product001	EUR
10	NULL	PPSNPL0	NULL	NULL	NULL	NULL	CustomerA	Product001	EUR



Unit_of_Measure_From	Date_Effective	Date_Expired	List_Price	Special_Price	Discount1	Discount2	Discount3	Discount4	Discount5
EA	2009-05-24	2009-05-24	1670.00000	NULL	0.10000	NULL	NULL	NULL	NULL
EA	2009-05-25	2009-12-31	1670.00000	NULL	0.05000	0.10000	NULL	NULL	NULL
EA	2010-01-01	2010-03-01	1650.00000	1550.00000	0.05000	0.10000	NULL	NULL	NULL
EA	2010-03-02	2010-08-21	1650.00000	NULL	0.05000	0.10000	NULL	NULL	NULL
EA	2010-08-22	2012-01-31	1650.00000	NULL	0.05000	0.07000	0.10000	NULL	NULL
EA	2012-02-01	2012-02-01	1650.00000	NULL	0.05000	0.07000	NULL	NULL	NULL
EA	2012-02-02	2012-02-02	1650.00000	NULL	0.08000	NULL	NULL	NULL	NULL
EA	2012-02-03	2015-12-31	1545.00000	NULL	0.08000	NULL	NULL	NULL	NULL
EA	2016-01-01	2016-02-29	1530.00000	1400.00000	0.08000	NULL	NULL	NULL	NULL
EA	2016-03-01	2039-12-31	1530.00000	NULL	0.08000	NULL	NULL	NULL	NULL

Table 18 - Pivot - Query Results

Now that the advanced pricing agreements are arranged this way, it’s possible to calculate the final price. As mentioned in the background and problem identification in chapter 1, whenever there is a special price and a discount in percentage occurring at the same time, the special price takes precedence over the discount by replacing the list price and only after is the discount in percentage applied.

For the example, the advanced pricing agreements state that any sale occurring between the 2010-01-01 and 1010-03-01 will have a special price of 1550€ and 2 discounts in percentage, 5% and 10%. The final price is calculated any of the following ways:

➔ [1550€ * (1 - 0,05)] * (1 - 0,1) = 1325,25€

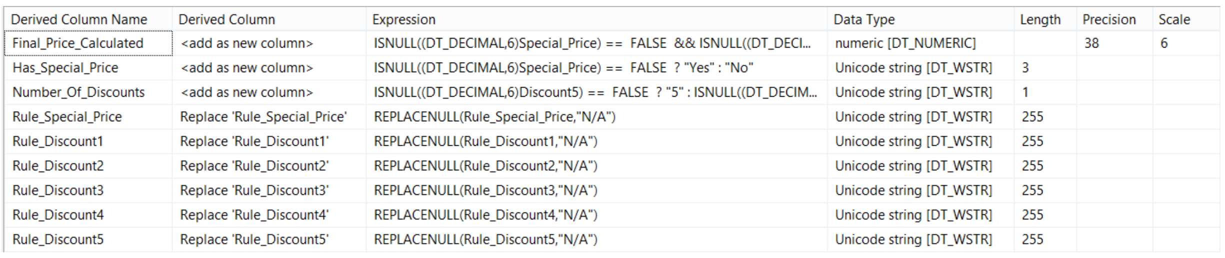
➔ [1550€ * (1 - 0,1)] * (1 - 0,05) = 1325,25€

The order in which the discounts in percentage are applied don't interfere with the final result.

If a sale occurred in 2012-02-01, the final price would be:

➔ [1650€ * (1 - 0,05)] * (1 - 0,07) = 1457,775€

Using the transformation from the query of Figure 145 as the main transformation of this phase, a SSIS Derived Column component was used to add the final price column as well as 2 other additional columns and replace null values for N/A (Not Applicable) to the existing rules for the discounts.



Derived Column Name	Derived Column	Expression	Data Type	Length	Precision	Scale
Final_Price_Calculated	<add as new column>	ISNULL((DT_DECIMAL,6)Special_Price) == FALSE && ISNULL((DT_DECI...	numeric [DT_NUMERIC]		38	6
Has_Special_Price	<add as new column>	ISNULL((DT_DECIMAL,6)Special_Price) == FALSE ? "Yes" : "No"	Unicode string [DT_WSTR]	3		
Number_Of_Discounts	<add as new column>	ISNULL((DT_DECIMAL,6)Discount5) == FALSE ? "5" : ISNULL((DT_DECIM...	Unicode string [DT_WSTR]	1		
Rule_Special_Price	Replace 'Rule_Special_Price'	REPLACENULL(Rule_Special_Price,"N/A")	Unicode string [DT_WSTR]	255		
Rule_Discount1	Replace 'Rule_Discount1'	REPLACENULL(Rule_Discount1,"N/A")	Unicode string [DT_WSTR]	255		
Rule_Discount2	Replace 'Rule_Discount2'	REPLACENULL(Rule_Discount2,"N/A")	Unicode string [DT_WSTR]	255		
Rule_Discount3	Replace 'Rule_Discount3'	REPLACENULL(Rule_Discount3,"N/A")	Unicode string [DT_WSTR]	255		
Rule_Discount4	Replace 'Rule_Discount4'	REPLACENULL(Rule_Discount4,"N/A")	Unicode string [DT_WSTR]	255		
Rule_Discount5	Replace 'Rule_Discount5'	REPLACENULL(Rule_Discount5,"N/A")	Unicode string [DT_WSTR]	255		

Figure 28 - Pivot - SSIS Derived Column

The final price is a calculated column and not original because it doesn't exist in the advanced pricing agreements data source. Therefore, by replicating this calculated measure, it will then be used to compare against the original final price measure from the invoiced sales data source for quality assurance, before being loaded to the presentation layer.

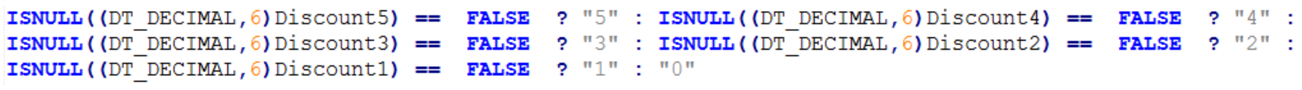
The SSIS expression for the final price column is represented in Figure 28 and uses the following logic to be calculated:

- If the Special Price column holds a value and the Discount1 does not, then the final price will be that Special Price;
- If the Special_Price holds a value, the Discount1 also has a value but the Discount2 does not, then the Final_Price_Calculated will be equal to Special_Price * (1 - Discount1). The same way, if there exists a Special_Price and Discount1 and Discount2 but not Discount3, then the price will be [Special_Price * (1 - Discount1)] * (1 - Discount2). The process continues up until Discount5;
- The same approach is done for whenever there is no Special_Price. If there is no Special_Price but there is a Discount1, Discount2, Discount3 and Discount4, then the Final_Price_Calculated calculated as being equal to ((((List_Price * (1 - Discount1)) * (1 - Discount2))) * (1 - Discount3)))) * (1 - Discount4))))).

The entire expression for the Final_Price_Calculated is found in the attachments in subchapter 7.5.2.

The flag Has_Special_Price was also added for filtering purposes on the presentation layer.

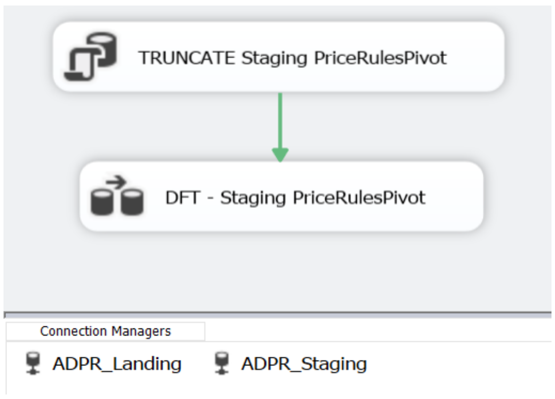
The Number_Of_Discounts column was also added and the full SSIS expression is presented in Figure 29.



```
ISNULL((DT_DECIMAL,6)Discount5)  ==  FALSE  ? "5" : ISNULL((DT_DECIMAL,6)Discount4)  ==  FALSE  ? "4" :
ISNULL((DT_DECIMAL,6)Discount3)  ==  FALSE  ? "3" : ISNULL((DT_DECIMAL,6)Discount2)  ==  FALSE  ? "2" :
ISNULL((DT_DECIMAL,6)Discount1)  ==  FALSE  ? "1" : "0"
```

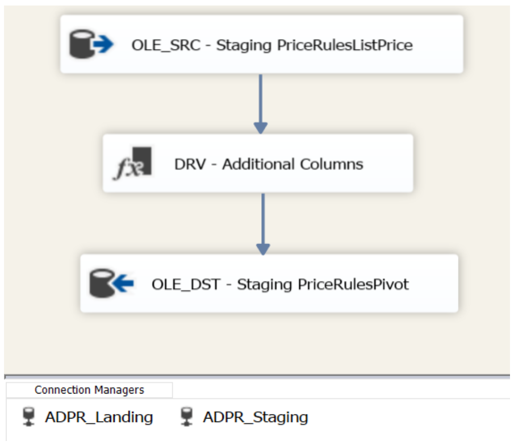
Figure 29 - SSIS Expression for Number_Of_Discounts

Finally, the Control Flow and Data Flow overall components are presented in Figures 30 and 31:



TRUNCATE Staging PriceRulesPivot
DFT - Staging PriceRulesPivot
Connection Managers
ADPR_Landing
ADPR_Staging

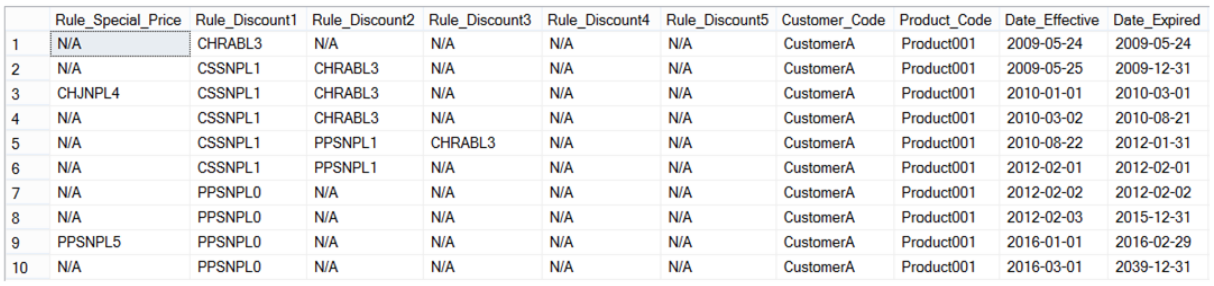
Figure 30 - Pivot - Control Flow



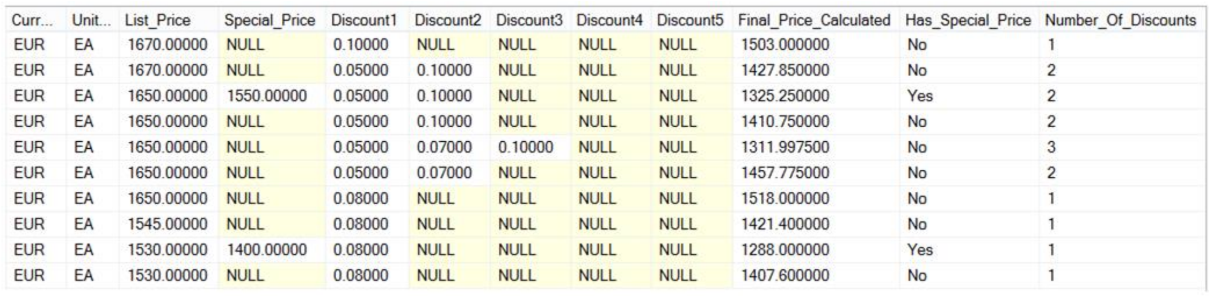
OLE_SRC - Staging PriceRulesListPrice
DRV - Additional Columns
OLE_DST - Staging PriceRulesPivot
Connection Managers
ADPR_Landing
ADPR_Staging

Figure 31 - Pivot - Data Flow

With the package completed and executed, and taking as example the data set from the date ranges, the final result is presented in the staging.PriceRulesPivot table in Table 19:



	Rule_Special_Price	Rule_Discount1	Rule_Discount2	Rule_Discount3	Rule_Discount4	Rule_Discount5	Customer_Code	Product_Code	Date_Effective	Date_Expired
1	N/A	CHRABL3	N/A	N/A	N/A	N/A	CustomerA	Product001	2009-05-24	2009-05-24
2	N/A	CSSNPL1	CHRABL3	N/A	N/A	N/A	CustomerA	Product001	2009-05-25	2009-12-31
3	CHJNPL4	CSSNPL1	CHRABL3	N/A	N/A	N/A	CustomerA	Product001	2010-01-01	2010-03-01
4	N/A	CSSNPL1	CHRABL3	N/A	N/A	N/A	CustomerA	Product001	2010-03-02	2010-08-21
5	N/A	CSSNPL1	PPSNPL1	CHRABL3	N/A	N/A	CustomerA	Product001	2010-08-22	2012-01-31
6	N/A	CSSNPL1	PPSNPL1	N/A	N/A	N/A	CustomerA	Product001	2012-02-01	2012-02-01
7	N/A	PPSNPL0	N/A	N/A	N/A	N/A	CustomerA	Product001	2012-02-02	2012-02-02
8	N/A	PPSNPL0	N/A	N/A	N/A	N/A	CustomerA	Product001	2012-02-03	2015-12-31
9	PPSNPL5	PPSNPL0	N/A	N/A	N/A	N/A	CustomerA	Product001	2016-01-01	2016-02-29
10	N/A	PPSNPL0	N/A	N/A	N/A	N/A	CustomerA	Product001	2016-03-01	2039-12-31



Curr...	Unit...	List_Price	Special_Price	Discount1	Discount2	Discount3	Discount4	Discount5	Final_Price_Calculated	Has_Special_Price	Number_Of_Discounts
EUR	EA	1670.00000	NULL	0.10000	NULL	NULL	NULL	NULL	1503.000000	No	1
EUR	EA	1670.00000	NULL	0.05000	0.10000	NULL	NULL	NULL	1427.850000	No	2
EUR	EA	1650.00000	1550.00000	0.05000	0.10000	NULL	NULL	NULL	1325.250000	Yes	2
EUR	EA	1650.00000	NULL	0.05000	0.10000	NULL	NULL	NULL	1410.750000	No	2
EUR	EA	1650.00000	NULL	0.05000	0.07000	0.10000	NULL	NULL	1311.997500	No	3
EUR	EA	1650.00000	NULL	0.05000	0.07000	NULL	NULL	NULL	1457.775000	No	2
EUR	EA	1650.00000	NULL	0.08000	NULL	NULL	NULL	NULL	1518.000000	No	1
EUR	EA	1545.00000	NULL	0.08000	NULL	NULL	NULL	NULL	1421.400000	No	1
EUR	EA	1530.00000	1400.00000	0.08000	NULL	NULL	NULL	NULL	1288.000000	Yes	1
EUR	EA	1530.00000	NULL	0.08000	NULL	NULL	NULL	NULL	1407.600000	No	1

Table 19 - Pivot - Full SSIS Package Results

3.3.3. Data Warehouse

The data warehouse is the final phase of the data preparation in the business intelligence framework presented in Figure 10 in the subchapter 3.2.2. Only the ETL of the dimension product table and the fact table for the advanced pricing agreements are presented in this subchapter since they both hold the main processes that are also used to some extent by the other data warehouse tables.

3.3.3.1. Dimension Product

This phase takes the data from the staging area and loads it into the dimension product. The objective is for every record of this dimension to be unique and list a product and the descriptive columns related to that product. The dimension product deals with slowly changing dimension attributes of type 1. As described by (Ralph Kimball & Ross, 2013, pp. 443-496), these attributes can be overwritten over time. To account for this, the Microsoft SSIS package for the dimension product was created using a mechanism to detect changes to the records of specific attributes and update them accordingly or insert new records. No indications were given by the pricing managers to maintain historical records that were changed. As such, any the changes overwrite the existing records.

SSIS Control Flow

The control flow is very rudimentary, containing only a data flow task. This task holds the necessary data flow for updating and inserting new records into the dimension in the data warehouse. There are two connections created to access the staging database and the data warehouse database in order to access the data from the staging and loading it into the data warehouse.

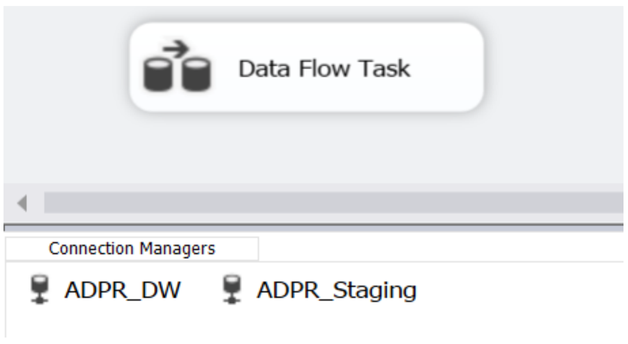


Figure 32 - Dimension Product - SSIS Control Flow

SSIS Data Flow

The data flow consists of the ETL transformations that happen from extracting the data from the staging.product table and inserting or updating it in the dwh.d_product table.

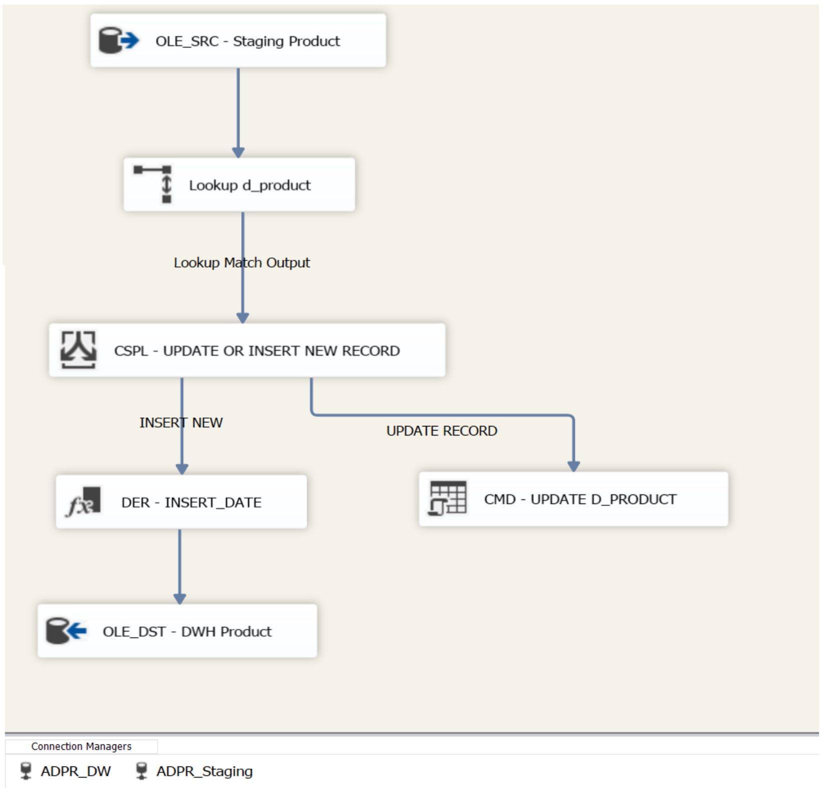


Figure 33 - Dimension Product - SSIS Data Flow

The first component is an OLE DB source, which is used to simply extract the data from the staging table staging.product. Within these columns, a column named HASH_TYPE1 is also accounted for. The Lookup_d_product component takes care of the lookup for the sk_product from the dimension d_product as well as the HASH_TYPE1 column that also exists in this dimension. The conditional split holds the logic to divide the data flow into the new records that will be inserted and the records that will be updated. For the new records, the data flow will go left and then insert the timestamp for logging purposes, before loading the records into the dimension product. For records that need to be updated, the data flow goes into the command component, which targets specifically the records that need to be updated and executes the updates.

Every component is explained in detail in subchapter 7.6 in the attachments, including the creation of dimension product with the use of the checksum function, responsible for formulating the HASH_TYPE1 column.

As a good practice, an inferred member containing all the unknown values for the respective columns is inserted in the dimension, as shown in Figure 34

Insert inferred members

```
SET IDENTITY_INSERT [dwh].[d_product] ON;

INSERT INTO [dwh].[d_product] ([sk_product],[Product_Code],[Product_Description],[Unit_Of_measure_Primary],[Product_Group],
                               [Price_Sub_Fran],[Price_Major],[Price_Region],[Sales_Rpt_08_Product_Discount_Group],[Branch_Plant])
VALUES ('0','-1','Unknown','Unknown','Unknown','Unknown','Unknown','Unknown','Unknown','Unknown');

SET IDENTITY_INSERT [dwh].[d_product] OFF;
```

Figure 34 - Dimension Product - Inferred members

3.3.3.2. Fact Table Advanced Pricing Agreements

The fact table for the advanced pricing agreements lists all the discounts for the combination of customer and product that are valid in a particular period of time. It's a temporal fact table, as described by (Mauri, 2012), since every record has a period that is valid from the date effective and valid to the date expired.

SSIS Control Flow

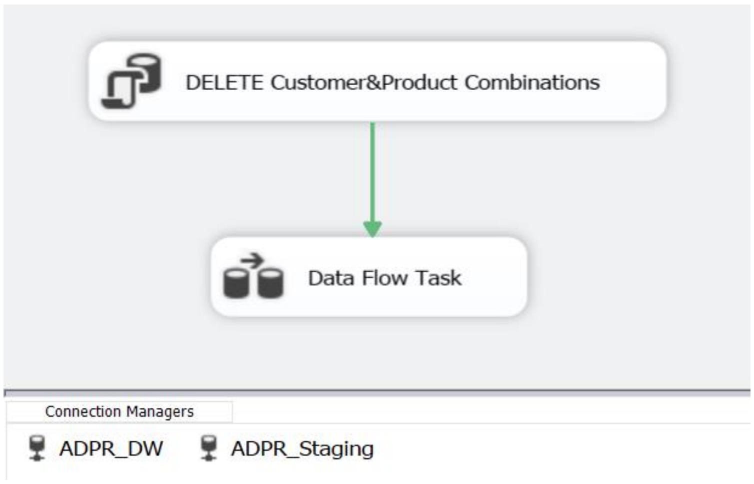
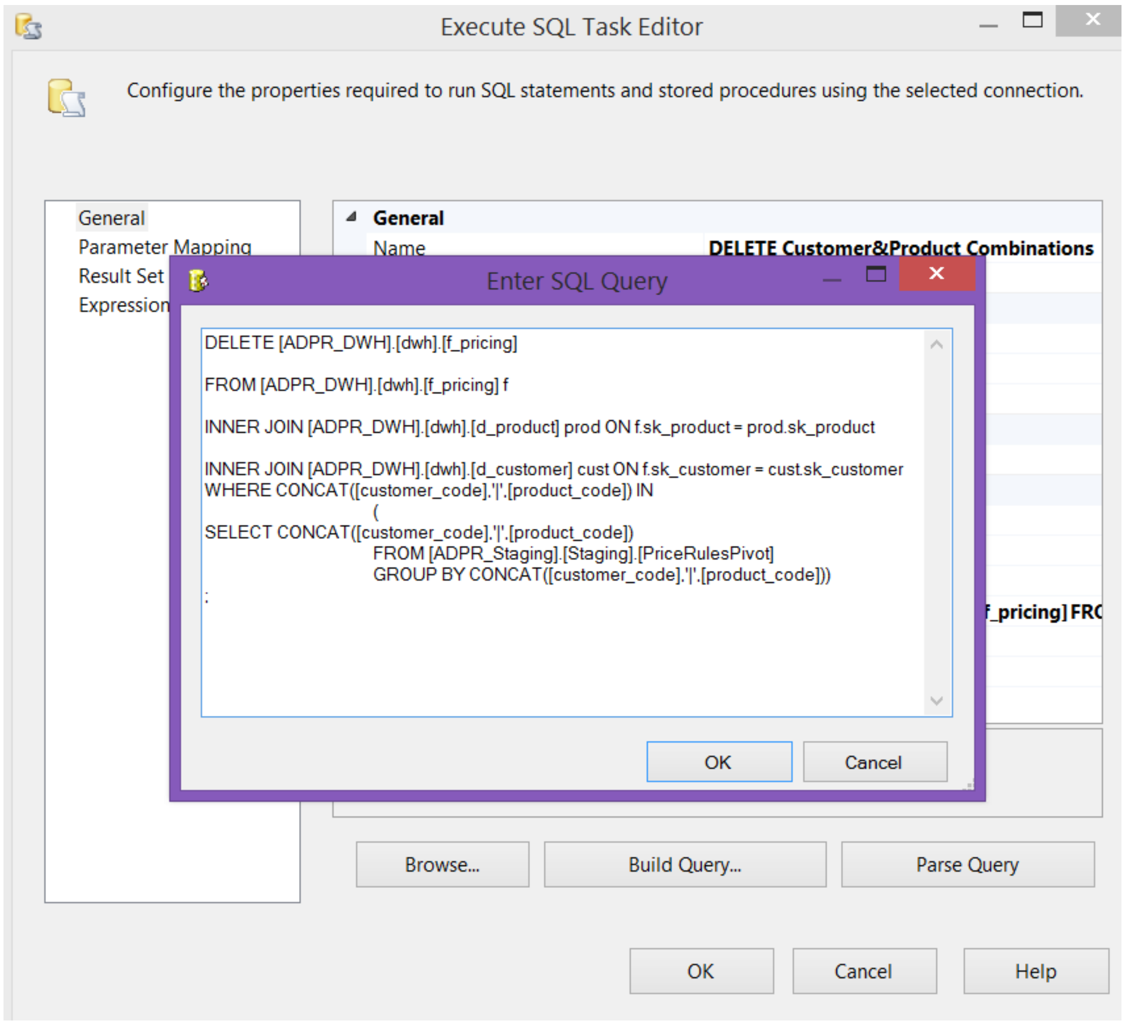


Figure 35 - SSIS Control Flow - Fact Table Advanced Pricing Agreements

Looking at the SSIS package for the data load into the fact table, the first component of the control flow deletes all the records from the fact table for the advanced pricing agreements that also exist in the staging.PriceRulesPivot. This is to prevent duplicate data. All other records that exist in the staging.PriceRulesPivot are new and require no deletion, and are, therefore, safe to load into the fact table.

By opening the component details, and analyzing the SQL query, all the combinations of Customer and Product that exist in the staging.PriceRulesPivot, regardless of dates, will be deleted from the fact table dwh.f_pricing. As instructed by the pricing managers, whenever new records are ingested, the new combinations of customer and product must also consider the old ones. Therefore, it's safe and effective to delete the records in this manner.



```
DELETE [ADPR_DWH].[dwh].[f_pricing]

FROM [ADPR_DWH].[dwh].[f_pricing] f

INNER JOIN [ADPR_DWH].[dwh].[d_product] prod ON f.sk_product = prod.sk_product

INNER JOIN [ADPR_DWH].[dwh].[d_customer] cust ON f.sk_customer = cust.sk_customer
WHERE CONCAT([customer_code],'|',[product_code]) IN
                        (
SELECT CONCAT([customer_code],'|',[product_code])
                        FROM [ADPR_Staging].[Staging].[PriceRulesPivot]
                        GROUP BY CONCAT([customer_code],'|',[product_code]))

;
```

Figure 36 - DELETE Customer&Product Combinations Component

The second component is the data flow task. This component is comprised of all the components that are responsible for accessing the data from the staging.PriceRulesPivot and loading it into the dwh.f_pricing.

Like in the SSIS package for the dimension product, there are two connections created to access both staging and data warehouse databases.

SSIS Data Flow

With the data transformations already completed at the level of the staging.PriceRulesPivot table, to create the fact table, the surrogate keys need to be added to link up the dimensions and form a star schema.

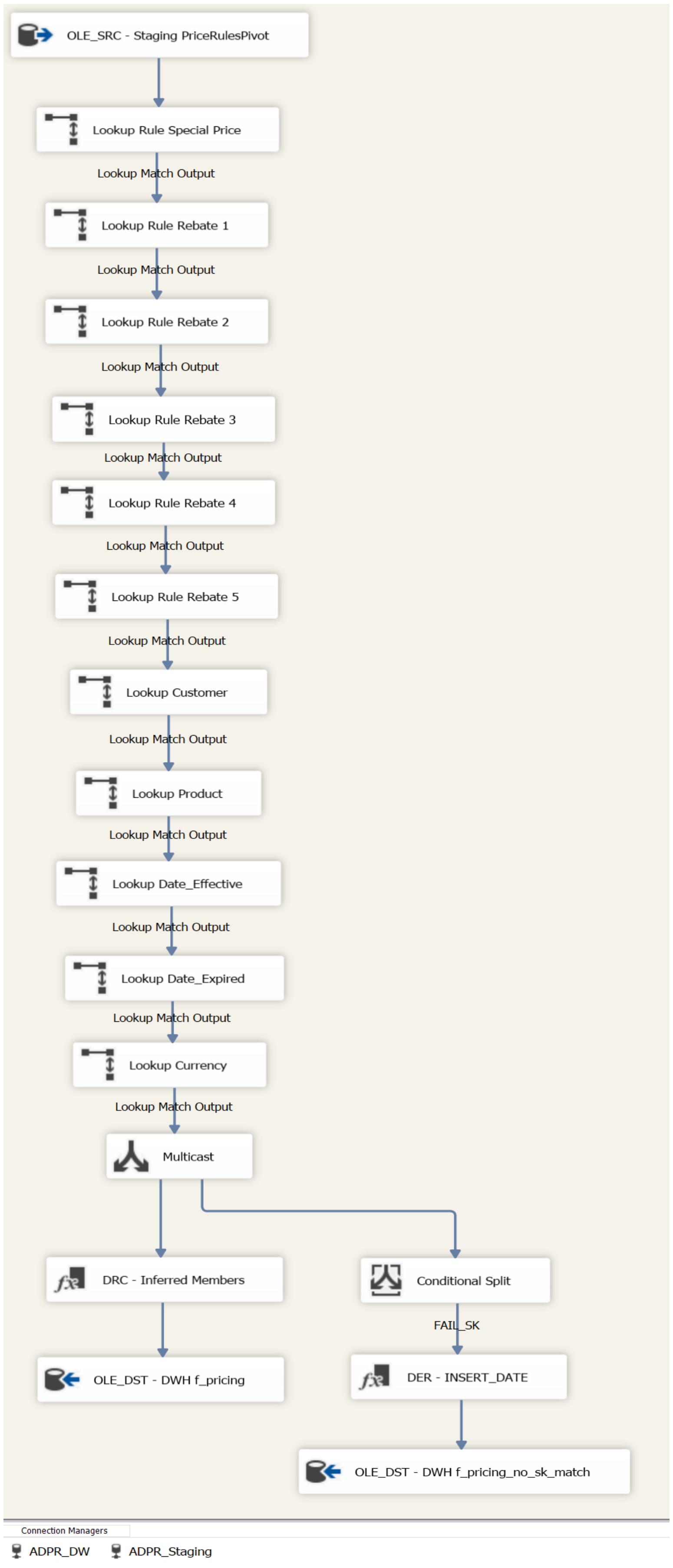


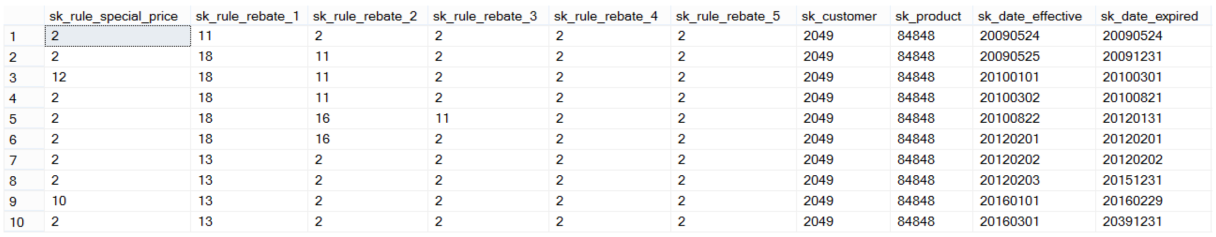
Figure 37 - SSIS Data Flow - Fact Table Advanced Pricing Agreements

The staging.PriceRulesPivot table is selected as the source of data for the first component. The following components lookup their respective dimensions, and in some cases more than once, such as the multiple discounts and date effective and date expired columns.

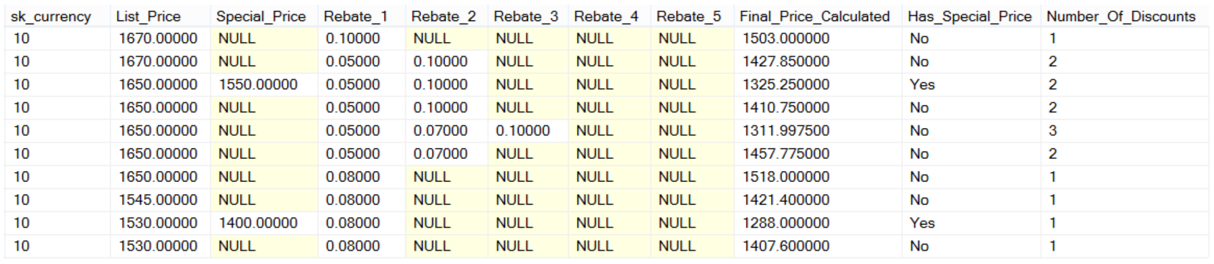
After the lookups, a conditional split is added so that records that fail to look up their surrogate key are inserted into the fact table with an inferred member -3, but also sent to a log table, which has the same structure as the fact table. This ensures referential integrity validation, since all surrogate keys that exist in the fact table must also exist in the dimensions. In the end, the data is inserted in the fact table dwh.f_pricing and the records with missing surrogate keys are also inserted in the log table with a timestamp, for logging purposes.

The component details and explanation can be found in the attachments in subchapter 7.7.

Following the main example case of this project, the final result after the data is inserted in the fact table is shown in Table 20:



	sk_rule_special_price	sk_rule_rebate_1	sk_rule_rebate_2	sk_rule_rebate_3	sk_rule_rebate_4	sk_rule_rebate_5	sk_customer	sk_product	sk_date_effective	sk_date_expired
1	2	11	2	2	2	2	2049	84848	20090524	20090524
2	2	18	11	2	2	2	2049	84848	20090525	20091231
3	12	18	11	2	2	2	2049	84848	20100101	20100301
4	2	18	11	2	2	2	2049	84848	20100302	20100821
5	2	18	16	11	2	2	2049	84848	20100822	20120131
6	2	18	16	2	2	2	2049	84848	20120201	20120201
7	2	13	2	2	2	2	2049	84848	20120202	20120202
8	2	13	2	2	2	2	2049	84848	20120203	20151231
9	10	13	2	2	2	2	2049	84848	20160101	20160229
10	2	13	2	2	2	2	2049	84848	20160301	20391231



sk_currency	List_Price	Special_Price	Rebate_1	Rebate_2	Rebate_3	Rebate_4	Rebate_5	Final_Price_Calculated	Has_Special_Price	Number_Of_Discounts
10	1670.00000	NULL	0.10000	NULL	NULL	NULL	NULL	1503.000000	No	1
10	1670.00000	NULL	0.05000	0.10000	NULL	NULL	NULL	1427.850000	No	2
10	1650.00000	1550.00000	0.05000	0.10000	NULL	NULL	NULL	1325.250000	Yes	2
10	1650.00000	NULL	0.05000	0.10000	NULL	NULL	NULL	1410.750000	No	2
10	1650.00000	NULL	0.05000	0.07000	0.10000	NULL	NULL	1311.997500	No	3
10	1650.00000	NULL	0.05000	0.07000	NULL	NULL	NULL	1457.775000	No	2
10	1650.00000	NULL	0.08000	NULL	NULL	NULL	NULL	1518.000000	No	1
10	1545.00000	NULL	0.08000	NULL	NULL	NULL	NULL	1421.400000	No	1
10	1530.00000	1400.00000	0.08000	NULL	NULL	NULL	NULL	1288.000000	Yes	1
10	1530.00000	NULL	0.08000	NULL	NULL	NULL	NULL	1407.600000	No	1

Table 20 - Fact Table Advanced Pricing Agreements - Final Result

3.3.3.3. Entity Relationship Diagram

The data warehouse model is comprised of 2 fact tables that share conformed dimensions. The fact tables are related and can be mutually accessed for more details. A sale from the dwh.f_transactions table can access the dwh.f_pricing table to search for valid discounts directly linked to that sale. In the same way, the dwh.f_pricing table can verify if sales occurred for a specific customer and product in any day in the dwh.f_transactions. As a result of this relation, according to (BAŞARAN, 2005, pp.43-44) the data warehouse design model is a fact constellation schema, as opposed to a galaxy schema, in which the fact tables are not directly related.

At an individualistic perspective, on one hand, dwh.f_transactions follows (Ralph Kimball & Ross, 2013, pp. 119-122 & 473-475) transactional type of fact table, where the atomic data is attributed to the daily transactions grain. On the other hand, dwh.f_pricing fits the criteria of Professor (Mauri, 2012) as a temporal type of fact table, where the atomic data represents all the discounts for a combination of customer and product that are valid between two dates.

At the dimension level, all data comes from the JDE Excel files from the Advanced Pricing Agreements. Nonetheless, all attributes of these dimensions are shared between the two fact tables, making them conformed dimensions.

The relation between the fact table and the dimensions is done through surrogate keys, rather than the natural keys present in the dimensions. This allows for better performance, since the surrogate keys use integers as data type, and complies with Kimball's dimensional modeling techniques to keep the records with a unique key in the event that a slowly changing attribute of type 2 is introduced in any of these dimensions.

Dimension dwh.d_rule only contains one column, aside from the surrogate key. This column contains the codes of the rules that are applied in the advanced pricing agreements. No additional descriptive columns were provided and are classified for the purpose of this project. The formula of how these rules are applied was not disclosed and the same goes for the order in which the rules are applied first. Because the order does not affect the calculation of the final price, it has no negative impact on the outcome of this project.

In theory, the List_Price from dwh.f_pricing should present the same value as the Unit_List_Price from dwh.f_transactions and the same goes for the Final_Price_Calculated and Unit_Price. These facts are not conformed facts, since these fact tables come from different sources and the advanced pricing agreements fact table was created to give visibility of the various layers of discounts that are automatically applied in the calculation of the prices that are present in the transactional fact table. Hence, there is a chance that these measures will differ. This can be validated by creating a data quality report, unifying these fact tables and exposing these records in case they exist.

The Entity Relationship Diagram shows the fact constellation schema and the relation between the two fact tables and the dimensions.

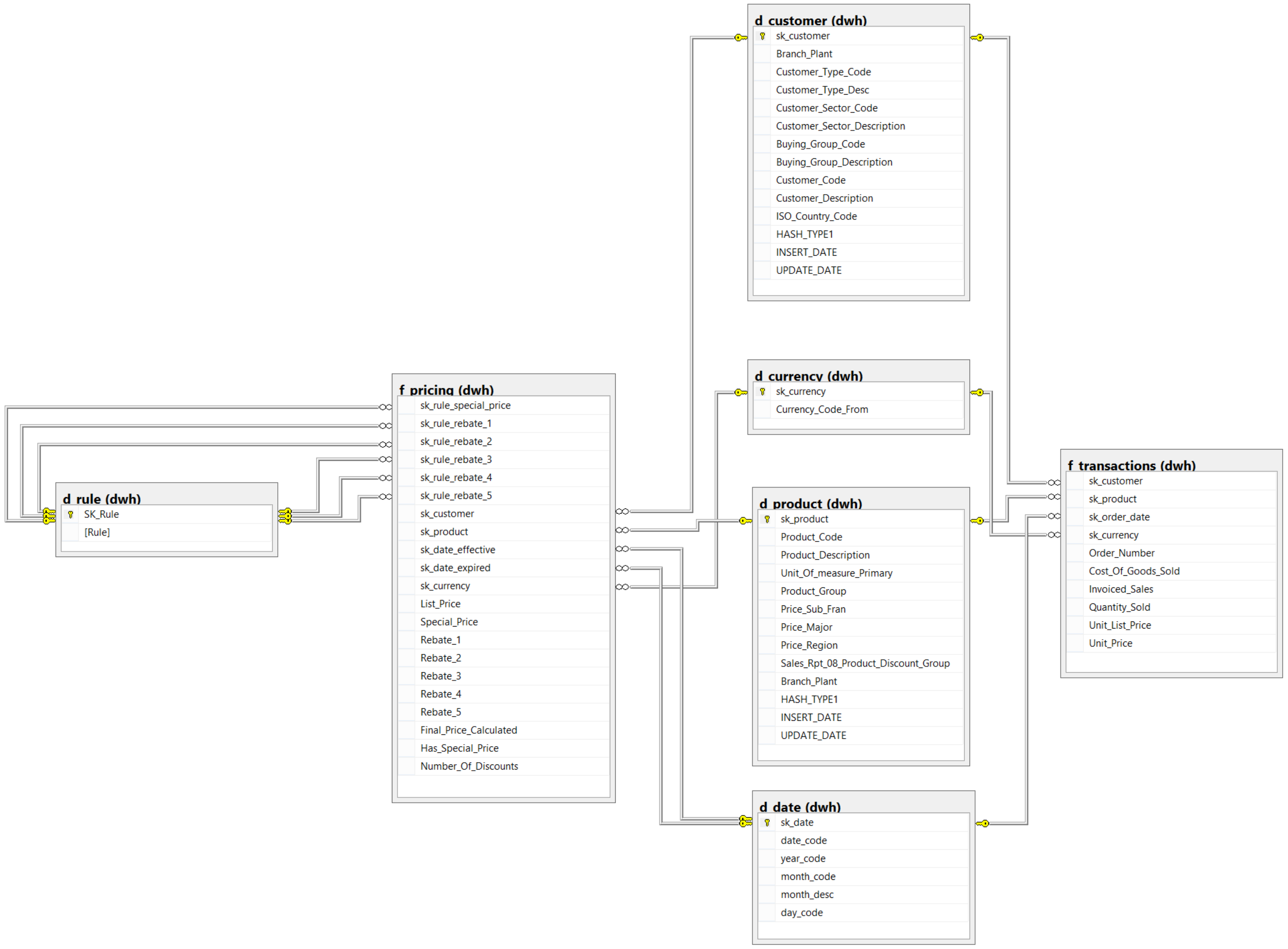


Figure 38 - Entity Relationship Diagram - Fact Constellation Schema

3.4. PRESENTATION LAYER

As mentioned in the methodology introduction in chapter 3 and in the business intelligence framework in subchapter 3.2.2, Qlik Sense was chosen by the business as the reporting tool to present the data in dashboards.

3.4.1. Data Load

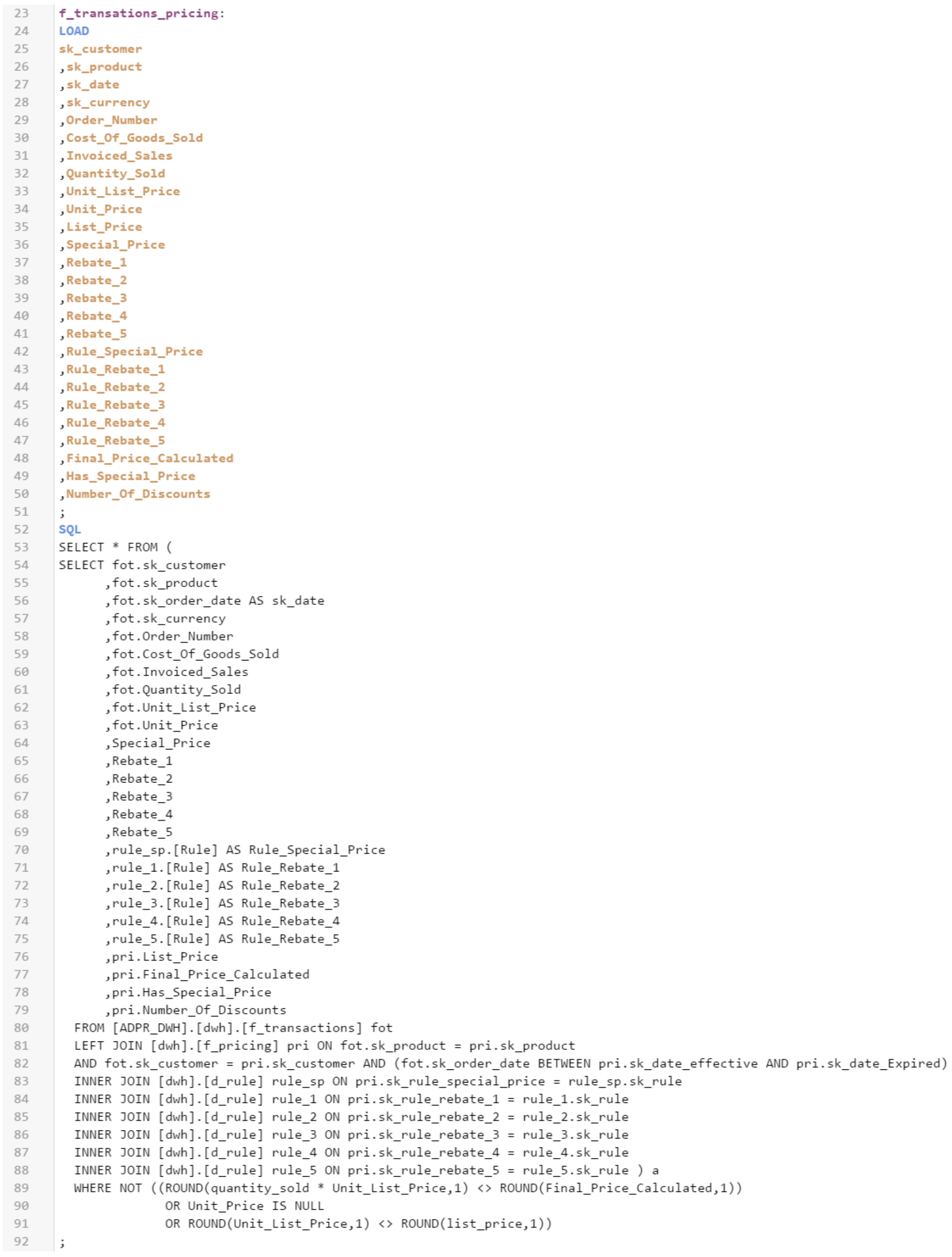
The first step to load the data from the data warehouse into a Qlik Sense app is to create a connection to the data warehouse in MS SQL Server. Once this is successfully done, the data can be loaded using the data load tool. Qlik Sense does not support fact constellation schemas, as such, the fact tables must be combined to form a single star schema. The primary goal is to analyze the sales that resulted from the advanced pricing agreements, so the granularity of the combined fact table will be at the transactional level. Each record will represent a sale accompanied by the various levels of discounts and/or special price. The load statement and the query used for the data load of this combined fact table is shown in Figure 39.

The query uses dwh.f_transactions as the source table to keep the granularity at the transactional level combined with a left join to retrieve the data from dwh.f_pricing. This table join takes effect for the same combination of product and customer, and where the order date occurred between a date effective and date expired of a given advanced pricing agreement, ensuring that all special price as and discount in percentage columns are true for that sale.

Since the dimension dwh.d_rule contains a single column for the price rules codes, and there can be one special price and 5 different discounts in percentage happening at the same time at best, an inner join for each of these columns is created to lookup the various columns that are used for the rules.

As mentioned before, the list_price and final_price_calculated measures are not conformed facts together with the unit_list_price and unit_price, and are, therefore, labelled differently. This is important to ensure quality assurance. The data in this app is loading records where the quantity_sold times the unit_list_price from the transactional fact table has to be equal to the final_price_calculated from the advanced pricing fact table, rounded to the decimal case. The same thing is applied to records that don’t have a unit_price or have a unit_list_price that is different from the list_price the advanced pricing agreements fact table.

The data load of the dimensions is comprised of the load of the columns followed by a query with a direct select statement, since no transformations need to occur.



```
f_transations_pricing:
LOAD
sk_customer
,sk_product
,sk_date
,sk_currency
,Order_Number
,Cost_Of_Goods_Sold
,Invoiced_Sales
,Quantity_Sold
,Unit_List_Price
,Unit_Price
,List_Price
,Special_Price
,Rebate_1
,Rebate_2
,Rebate_3
,Rebate_4
,Rebate_5
,Rule_Special_Price
,Rule_Rebate_1
,Rule_Rebate_2
,Rule_Rebate_3
,Rule_Rebate_4
,Rule_Rebate_5
,Final_Price_Calculated
,Has_Special_Price
,Number_Of_Discounts
;
SQL
SELECT * FROM (
SELECT fot.sk_customer
      ,fot.sk_product
      ,fot.sk_order_date AS sk_date
      ,fot.sk_currency
      ,fot.Order_Number
      ,fot.Cost_Of_Goods_Sold
      ,fot.Invoiced_Sales
      ,fot.Quantity_Sold
      ,fot.Unit_List_Price
      ,fot.Unit_Price
      ,Special_Price
      ,Rebate_1
      ,Rebate_2
      ,Rebate_3
      ,Rebate_4
      ,Rebate_5
      ,rule_sp.[Rule] AS Rule_Special_Price
      ,rule_1.[Rule] AS Rule_Rebate_1
      ,rule_2.[Rule] AS Rule_Rebate_2
      ,rule_3.[Rule] AS Rule_Rebate_3
      ,rule_4.[Rule] AS Rule_Rebate_4
      ,rule_5.[Rule] AS Rule_Rebate_5
      ,pri.List_Price
      ,pri.Final_Price_Calculated
      ,pri.Has_Special_Price
      ,pri.Number_Of_Discounts
  FROM [ADPR_DWH].[dwh].[f_transactions] fot
  LEFT JOIN [dwh].[f_pricing] pri ON fot.sk_product = pri.sk_product
  AND fot.sk_customer = pri.sk_customer AND (fot.sk_order_date BETWEEN pri.sk_date_effective AND pri.sk_date_Expired)
  INNER JOIN [dwh].[d_rule] rule_sp ON pri.sk_rule_special_price = rule_sp.sk_rule
  INNER JOIN [dwh].[d_rule] rule_1 ON pri.sk_rule_rebate_1 = rule_1.sk_rule
  INNER JOIN [dwh].[d_rule] rule_2 ON pri.sk_rule_rebate_2 = rule_2.sk_rule
  INNER JOIN [dwh].[d_rule] rule_3 ON pri.sk_rule_rebate_3 = rule_3.sk_rule
  INNER JOIN [dwh].[d_rule] rule_4 ON pri.sk_rule_rebate_4 = rule_4.sk_rule
  INNER JOIN [dwh].[d_rule] rule_5 ON pri.sk_rule_rebate_5 = rule_5.sk_rule ) a
  WHERE NOT ((ROUND(quantity_sold * Unit_List_Price,1) <> ROUND(Final_Price_Calculated,1))
              OR Unit_Price IS NULL
              OR ROUND(Unit_List_Price,1) <> ROUND(list_price,1))
;
```

Figure 39 - Data Load for f_transactions_pricing

Once the data has been loaded, the data model can be accessed.

3.4.2. Data Model

The data model displays the relation between the combined fact table and the dimensions to provide contextual descriptions of each sale record. The tables are linked through the surrogate key columns in a similar way to the data warehouse.

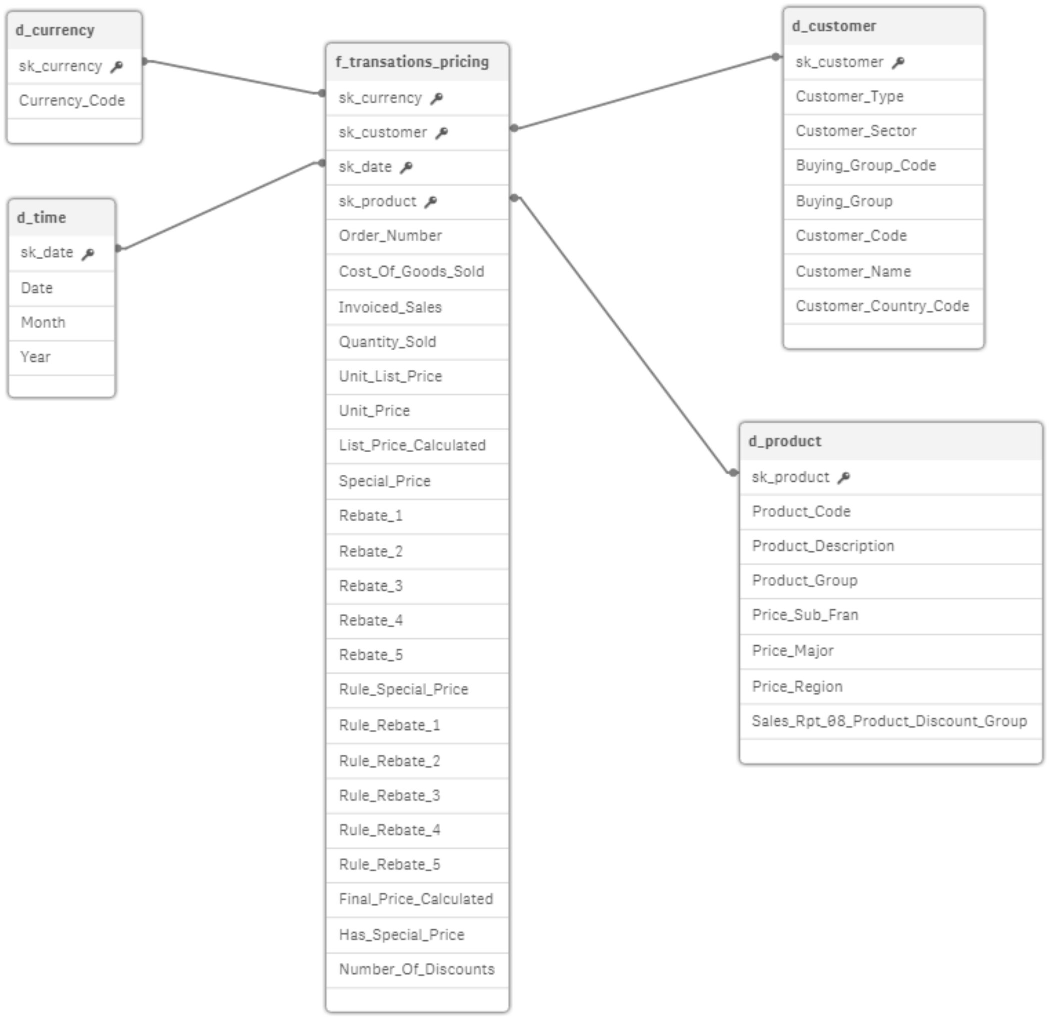
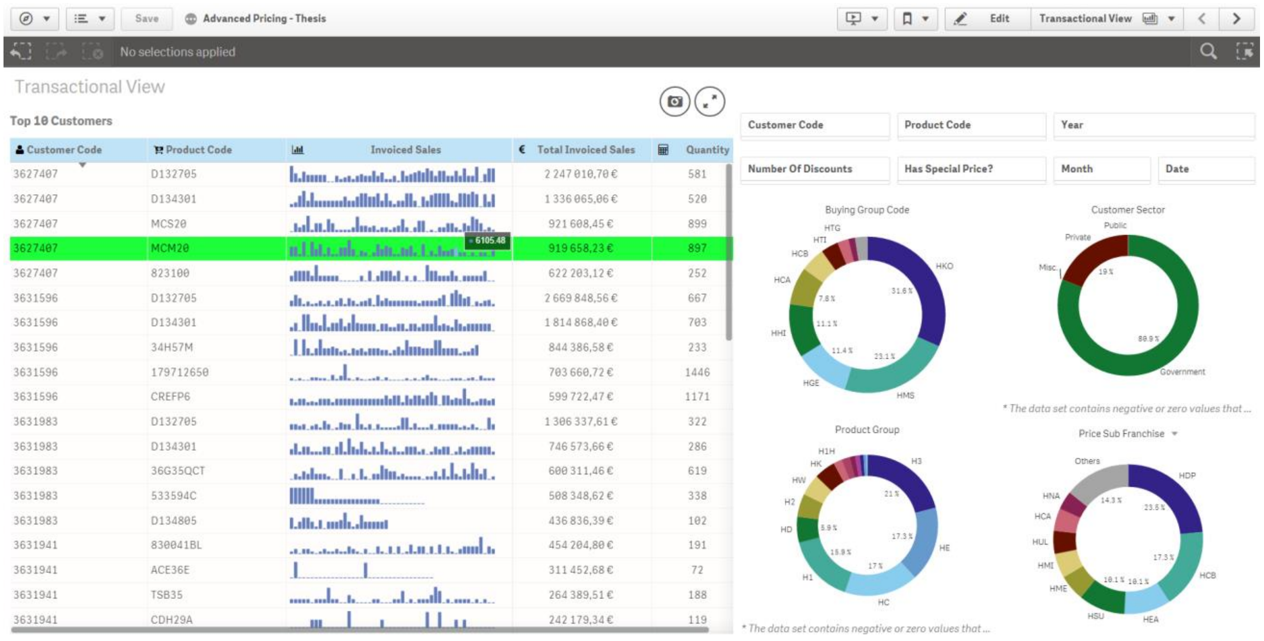


Figure 40 - Qlik Sense - Data Model

The dimension d_time has also been created differently with a logic to include calculated time periods such as CYTD and PYTD. Since it serves no significant purpose for this specific app to compare sales with advanced pricing agreements, it’s has been left on the side for other analysis in apps that may require these time periods. As such, this logic is attached in the attachment section in subchapter 7.8.

3.4.3. Dashboard - Transactional View sheet

In the front end, the business user can access 2 sheets within the app. The first sheet is the Transactional View, as shown in Figure 41:



Top 10 Customers

Customer Code	Product Code	Invoiced Sales	Total Invoiced Sales	Quantity
3627407	D132705		2 247 010,70 €	581
3627407	D134301		1 336 065,06 €	520
3627407	MCS20		921 608,45 €	899
3627407	MCM20	6105.48	919 658,23 €	897
3627407	823100		622 203,12 €	252
3631596	D132705		2 669 848,56 €	667
3631596	D134301		1 814 868,40 €	703
3631596	34H57M		844 386,58 €	233
3631596	179712650		703 660,72 €	1446
3631596	CREFP6		599 722,47 €	1171
3631983	D132705		1 306 337,61 €	322
3631983	D134301		746 573,66 €	286
3631983	36G35QCT		600 311,46 €	619
3631983	533594C		508 348,62 €	338
3631983	D134805		436 836,39 €	102
3631941	830041BL		454 204,80 €	191
3631941	ACE36E		311 452,68 €	72
3631941	TSB35		264 389,51 €	188
3631941	CDH29A		242 179,34 €	119

Figure 41 - Transactional View sheet

This sheet is focused on delivering short-term goals to mid-level management, using (Larson, 2016, pp. 13-21) pyramid for attributing specific goals. The visualizations are presented with summarized data with the ability to drill-down for more details.

The list table entitled "Top 10 Customers" is a Vizlib Qlik Sense extension that was installed to add the minichart feature that is present in the "Invoiced Sales" column. This allows for micro analysis within a table itself by hovering the mouse on top of each bar to get the value. The table shows the top 10 customers with the highest invoiced sales, and for each customer the top 5 products can be observed. Additionally, the quantity sold is also presented.

On the right side, 4 pie charts are presented with the most relevant category information regarding invoiced sales formed on each visualization. The bottom right pie chart supports alternate dimensions by clicking on the arrow, which can be chosen from Price Sub franchise, Price Major and Price Region.

Lastly, the filter panes can be found on the top right corner, allowing for a more targeted selection of data, such as the sale data for a specific customer and product information for a certain year that resulted from 3 discounts valid at the same time.

All the visualizations are dynamic and will adjust to any filter being applied.

By clicking on top of the minicharts or on the total invoiced sales, the combination of customer and product are automatically applied as a filter and the Pricing Analysis sheet appears with the more detailed data.

3.4.4. Dashboard - Pricing Analysis sheet

The second sheet is directed specifically for in-dept analysis and is presented in Figure 42. It's aimed at pricing managers who deal with day-to-day operational goals that need access to detailed-level data regarding specific customers and products.

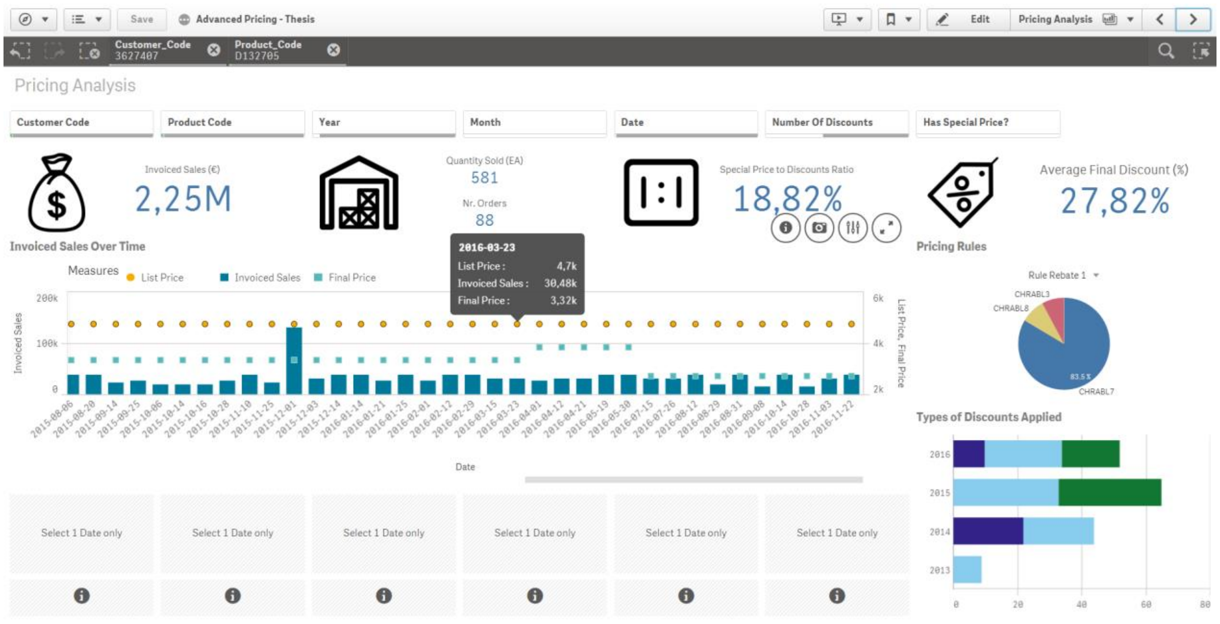


Figure 42 - Pricing Analysis sheet - General

At the very top, the filters can be found. Specific Customers and Products can be selected as well as a particular year, month or even date. The last 2 filters allow to select data that only contains a certain number of discounts and another for records with special price.

There are 5 measures at the second layer of the dashboard. The total of invoiced sales and the total of the quantity sold in units are simply measures that require only a sum() aggregate function. The number of orders placed uses a count(distinct) aggregate function in order to correctly group the records by the individual orders, which may be comprised of multiple records, depending on where a customer purchased more than one product on the same order. Finally, the last 2 are calculated measures. The Special Price to Discounts Ratio is formulated to give the ratio of a special price in regard to the total number of discounts, and the average final discount in percentage compares the final price to the list price and presents the difference in percentage.

The Qlik Sense expression to calculate the Special Price to Discounts Ratio is the following:
(COUNT(Special_Price)/(COUNT(Special_Price) + COUNT(Rebate_1) + COUNT(Rebate_2) + COUNT(Rebate_3) + COUNT(Rebate_4) + COUNT(Rebate_5)))

The Qlik Sense expression to calculate the Average Final Discount (%) is:

1 - (sum(Final_Price_Calculated) / sum(List_Price))

On the Invoiced Sales Over Time bar chart, the horizontal axis shows the invoiced sales dates represented by the rectangular vertical bars, and the list price and final price, in green squares and yellow circles, respectively. Note that the Final Price is the measure Final_Price_Calculated, but renamed for a more appropriate business name. On the vertical Axis, the values on the left provide the numerical representation for the invoiced sales and the values on the right are scaled for both prices. This is the main visualization of the dashboard. It allows to track the invoiced sales trends over time while providing a visual representation of the difference in list price to the final price, providing insights that help analyze the reasons behind increases and falls.

The Pricing Rules pie chart provides the distribution of the rules that are being applied for the discount that is selected, from the special price through rebates 1 to 5. As an example, Figure 42 shows that by selecting Rule Rebate 1 on the pie chart, the rule with the code CHRABL7 is being applied to 83,5% of the total invoiced sales for the selected customer and product. The purpose is to be able to allocate the invoiced sales to the rules applied to understand the share and importance each rule holds.

Right below is a horizontal stacked bar chart to provide insights on the number of times each type of discount being applied. While hovering the mouse on top of the 2016 bar, it shows that there are 10 invoiced sales with a special price, 24 with a discount 1 in percentage or rebate 1 for short, and 18 with a rebate 2.

Both previous visualizations have a legend that only appears when the visualizations are big enough. Therefore, by expanding these visualizations, the legends can be visualized. Another way to obtain the details of these visualizations is by hovering the mouse on top of the intended data.

By selecting a date such as 2016-07-26, as shown in Figure 43, the Invoiced Sales Over Time highlights the respective bar and the 6 measures at the bottom are enabled because of the following calculation condition used in the data handling section of the visualization:

=GetSelectedCount(Date)=1

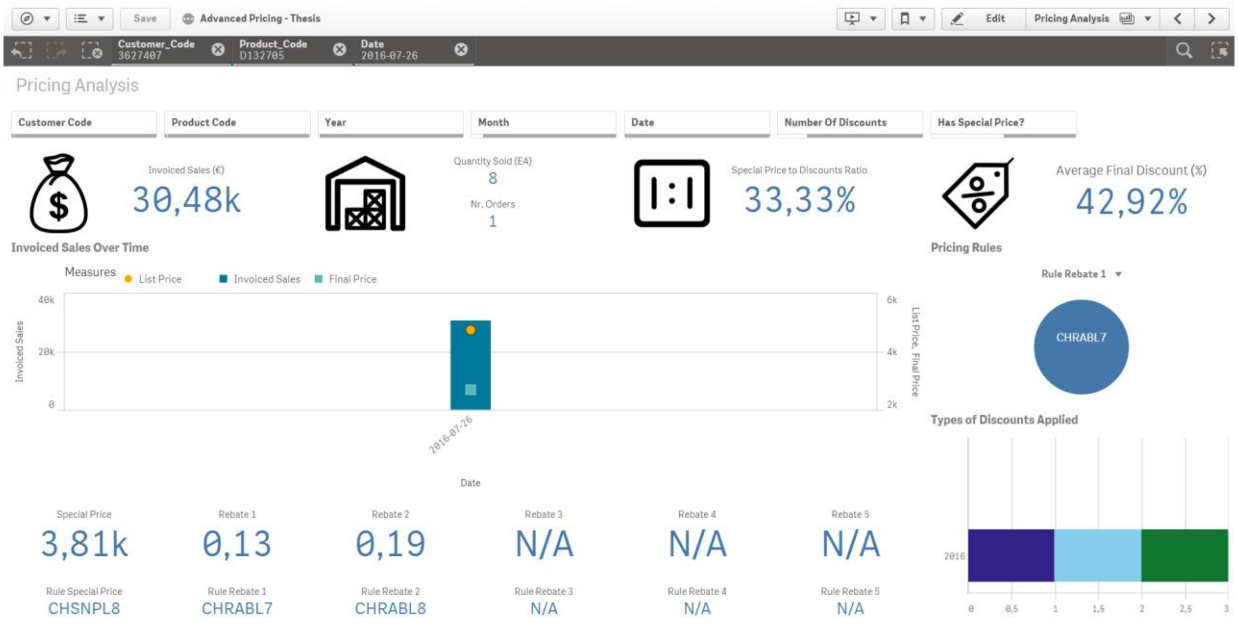


Figure 43 - Pricing Analysis sheet - Targeted

With this condition, when selecting a single date, it is possible to obtain the information regarding all the discounts that are being applied, their types, amounts and rules, providing the pricing managers with the targeted information that was requested.

By selecting a specific date, the corresponding special price value and rule are shown if they exist. The expression used to calculate the special price is the following:

Special Price: IF(ISNULL(Special_Price) = '-1', 'N/A', MIN(Special_Price))

Using the same logic, the rule also uses a very similar expression:

Rule Special Price (Rule): IF(ISNULL(Special_Price) = '-1', 'N/A', Rule_Special_Price)

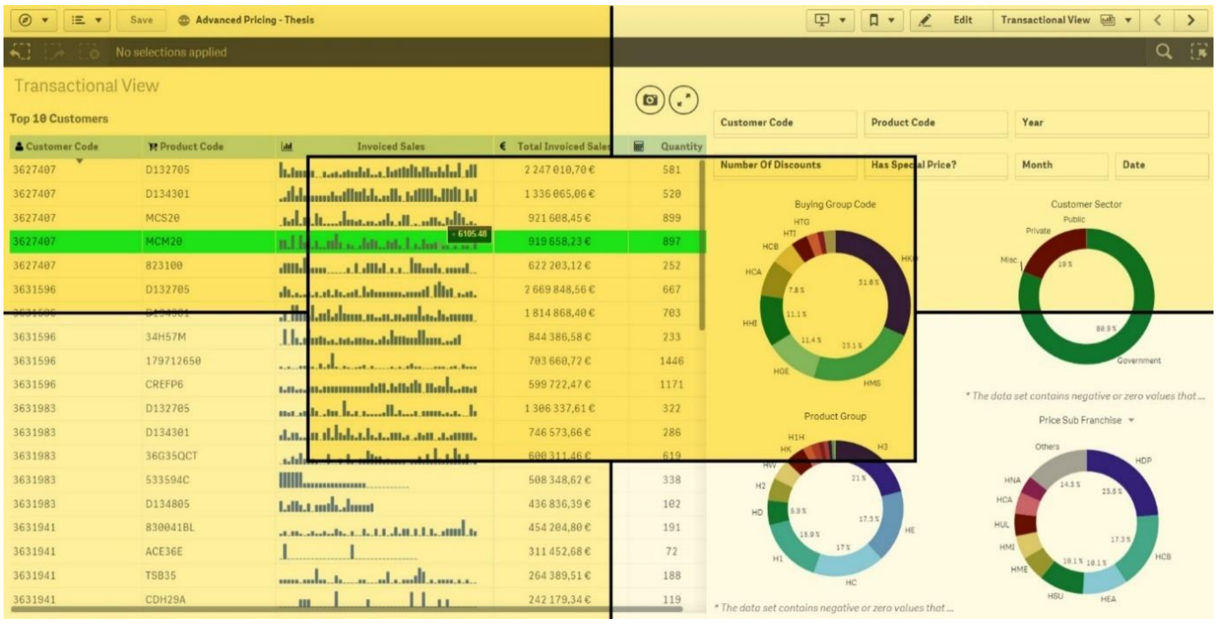
Rebates 1 through 5 have the exact same expressions, but use their own measures and dimensions.

4. RESULTS

With the business intelligence framework completed, although the majority of the work takes place at the data preparation, the front-end of the presentation layer displays the end results. It is through the efficient design of the dashboards, the navigation, the capacity to filter the desired data, and the quickness of the visualizations to load that weighs the quality of the results. Only meeting the business requirements by delivering the intended data is not enough, the dashboards must be appealing to the user, and built in such a way that no matter the complexity behind the process leading to the data presentation, the practical use of accessing data must be user-friendly.

The data has been covered in both chapters 3.3.3 from the data warehouse perspective and chapter 3.4 from the Qlik Sense app that was created to combine the invoiced sales with the layers of advanced pricing agreements.

Analyzing the degrees of visual emphasis on the different regions of the dashboard, and by marking the Transactional View sheet using the same layout suggested by (Few, 2006, pp. 80-94), the 5 regions are presented as follows:



Transactional View

Top 10 Customers

Customer Code	Product Code	Invoiced Sales	Total Invoiced Sales	Quantity
3627407	D132705		2 247 010,70 €	581
3627407	D134301		1 336 065,06 €	520
3627407	MCS20		921 608,45 €	899
3627407	MCM20		919 658,23 €	897
3627407	823100		622 203,12 €	252
3631596	D132705		2 669 848,56 €	667
3631596	D134301		1 814 868,40 €	703
3631596	34H57M		844 386,58 €	233
3631596	179712650		703 660,72 €	1446
3631596	CREFP6		599 722,47 €	1171
3631983	D132705		1 306 337,61 €	322
3631983	D134301		746 573,66 €	286
3631983	36G35QCT		600 311,46 €	619
3631983	533594C		508 348,62 €	338
3631983	D134805		436 836,39 €	102
3631941	830041BL		454 204,80 €	191
3631941	ACE36E		311 452,68 €	72
3631941	TSB35		264 389,51 €	188
3631941	CDH29A		242 179,34 €	119

Figure 44 - Transactional View - Degrees of Visual Emphasis

The center and top left rectangles represent the emphasized regions and hold most of the more important information in the transactional view. The top layer of the Top 10 Customers visualization, where the total values are found as well as the data from the top 1 customer and 5 products are within the limits of this area. Most of the minicharts are also located inside the emphasized areas as well as the most important pie chart, which is the buying groups representing their shares of the invoiced sales.

The bottom half of the Top 10 Customers as well as the filters are located in the Neither emphasized nor de-emphasized areas, considering that these areas are the second most important locations to turn to.

The whole dashboard must be occupied for efficiency, so even the non-emphasized areas should hold information, although the least important. The pie chart located at the bottom right can be found on a region that is de-emphasized as it is the least important visualization, according to the business users.

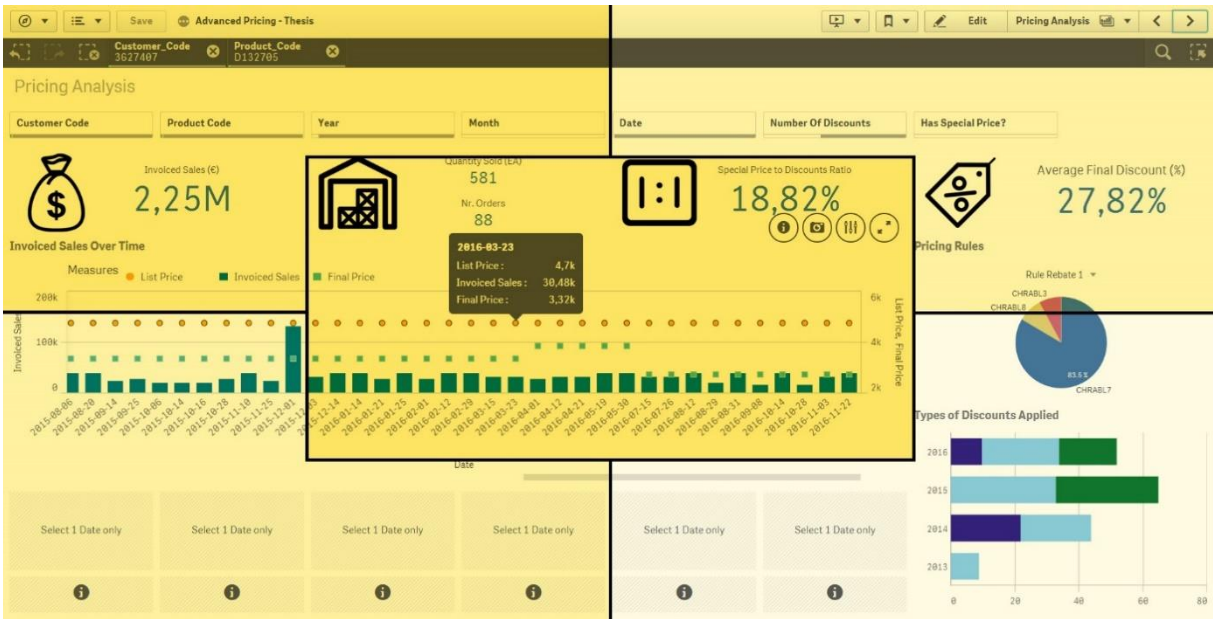


Figure 45 - Pricing Analysis - Degrees of Visual Emphasis

On the Pricing Analysis sheet, the most important measures as well as the main body of the Invoiced Sales Over Time visualization are located inside the 2 emphasized regions.

On the bottom layer of the dashboard, the first 4 discount rules and values are presented in the area that is neither emphasized or de-emphasized, since these layers of discounts are only accessed after a date has been selected, they are considered a secondary access priority.

The last 2 discount rules and values are located in a de-emphasized area, as very rarely are there more than 3 discounts in percentage valid in the same period. On the same region, the visualization displaying the Types of Discounts Applied and most of the Pricing Rules pie chart are also located here. Since most of the visualizations are very important in this sheet and because of the structural design, these visualizations although not worthy of being de-emphasized, have no other place in the dashboard.

Overall, both sheets comply closely with the design principles for usability covered by (Few, 2006, pp. 138-146), by:

- co-locating visualizations that serve the same function in an ordered and organized manner to allow for meaningful comparisons, such as keeping the 4 pie charts together on the transactional view sheet as well as the total measures and discount rules in the Pricing Analysis sheet;

- Not varying means of display on similar visualizations and functionalities to maintain consistency across both sheets;

- Making the dashboards aesthetically appealing, which helps the business users to enjoy the dashboards, feel more relaxed and absorb the insights by creating an overall better response. Using the same soft colors tones, keeping the data presentation simple and straight;

- Making the dashboard interaction-oriented so that the business user can drill-down on invoiced sales to discover the layers of discounts applied. Providing filters and enabling the user to directly select data on the visualizations to slice the data to their own requirement;

- Testing the dashboards for usability to ensure the look and feel meet the expectations, as well as providing a prototype before the final product goes into development to easily allow for changes. The objective of this project report serves as a prototype for a more extensive development.

5. CONCLUSIONS

This project report aimed to introduce a business intelligence framework for advanced pricing agreements. More specifically, to build a solution that would give visibility to the pricing managers on the various layers of discounts that took part in the transactions that followed those advanced pricing agreements.

Along the way, many challenges were faced and provided with a solution to get to the intended result. The most demanding and troublesome obstacle was to find a way to separate multiple discounts occurring at the same time, but with different date ranges and different list prices with date ranges of their own, into common date ranges. The solution required the creation of a new method that had to be fully tested to make up the formula for solving this multiple date range problem. This was achieved and demonstrated in subchapter 3.3.2.2 coupled with full details in subchapter 7.3 and the complete testing in subchapter 7.4. This formula can be applied to any scenario with overlapping date ranges and with or without the list price logic, if required.

At the data warehouse level, every record from the advanced pricing agreements fact table holds a date effective and a date expired, indicating all the discounts that are valid during that period of time, and this is performed for every customer and product combination. Because every record holds a date range, this fact table is categorized as a temporal fact table. This is important to study because the academic coverage on this topic is very scarce and this type of fact table can play a very vital role in relaying essential information to business users in similar businesses and fields. In advanced pricing agreements, it allows for the records to be easily combined with the invoiced sales in order to provide all the necessary details for the pricing managers to enable them to make a targeted analysis and facilitate future price adjustments.

At the end, two interactive dashboards were developed in Qlik Sense, which present the combined data from the two fact tables in order to provide insights relevant to mid-level managers as well as pricing managers.

As for the future of this project, it is important to remember that this is a prototype served for a bigger project intended by the client. Many other developments can take place to enrich the data warehouse and presentation layer. For this reason, the project was implemented by following Kimball's bottom's up approach to establish a final data mart and leave room for additional work to be integrated. Case in point, the dimension Rule, at this stage, is very limited to simple codes. As such, additional descriptive columns can be added for a more detailed understanding. More information found in other tables such as the sequence in which the rules are applied can also enhance the reasoning behind the implementation of those rules. Same logic applies to the dimension Currency, where more currencies can be added as well as additional tables containing data regarding exchange rates. Other examples of further developments can be attributed to additional Qlik Sense apps to serve the purpose of data quality to assure that there isn't anything missing or incorrect in the data warehouse. Another example would be to create an app solely with advanced pricing agreements data to give visibility on all the agreements that exist in the fact table and not only those linked to invoiced sales, as some hidden agreements that did not incur in sales may play an important part in the analysis.

6. BIBLIOGRAPHY

Baird, N. (2017). Dynamic vs. Personalized Pricing. Retrieved from https://www.rsrresearch.com/research/dynamic-vs-personalized-pricing

BAŞARAN, B. P. (2005). a Comparison of Data Ware House Design Models. *Analyzer*, 90. https://doi.org/10.1.1.463.1611

Chen, H. (Allan), & Rao, A. R. (2007). When Two Plus Two Is Not Equal to Four: Errors in Processing Multiple Percentage Changes. *Journal of Consumer Research*, *34*(3), 327–340. https://doi.org/10.1086/518531

Chen, H., Chiang, R. H. L., & Storey, V. C. (2012). Business Intelligence and Analytics: From Big Data To Big Impact. *Mis Quarterly*, *36*(4), 1165–1188. https://doi.org/10.1145/2463676.2463712

Chen, S. F. S., Monroe, K. B., & Lou, Y. C. (1998). The effects of framing price promotion messages on consumers' perceptions and purchase intentions. *Journal of Retailing*, *74*(3), 353–372. https://doi.org/10.1016/S0022-4359(99)80100-6

Competera. (2019). Dynamic Pricing Engine Software: Your Key to Optimal Value Offering. Retrieved from https://competera.net/resources/articles/dynamic-pricing-engine

Elegido, J. M. (2012). The Ethics of Price Discrimination. *Business Ethics Quarterly*. https://doi.org/10.5840/beq201121439

Few, S. (2006). Information Dashboard Design. *The Effective Visual Communication of Data Sebastopol*, (January), 223. https://doi.org/10.1017/S0021849904040334

Grewal, D., Ailawadi, K. L., Gauri, D., Hall, K., Kopalle, P., & Robertson, J. R. (2011). Innovations in Retail Pricing and Promotions. *Journal of Retailing*, *87*, S43–S52. https://doi.org/10.1016/j.jretai.2011.04.008

Isabella, G., Pozzani, A. I., Chen, V. A., & Perfi Gomes, M. B. (2012). Influence of Discount Price Announcements on Consumer'S Behavior. *Influencia De Los Avisos De Descuento Sobre El Comportamiento De Los Consumidores.*, *52*(6), 657–671. https://doi.org/10.1086/380287

JD Edwards. (2020). JDE Tables Factor Value. Retrieved from http://jderef.com/?schema=812&column=FVTR

Kimball, R., & Caserta, J. (2004). *The Data Warehouse ETL Toolkit: Practical Techniques for Extracting, Cleaning Conforming, and Delivering Data*. *Wiley, 1st Edition*. https://doi.org/10.1017/CBO9781107415324.004

Kimball, Ralph, & Ross, M. (2013). *The Data Warehouse Toolkit, The Definitive Guide to Dimensional Modeling*. *The Data Warehouse Toolkit, The Definitive Guide to Dimensional Modeling* (3rd ed.). https://doi.org/10.1145/945721.945741

Larson, B. (2016). *Delivering Business Intelligence with Microsoft SQL Server 2016*. (McGraw-Hill Education, Ed.) (4th editio).

Lipovetsky, S. (2011). Pricing Models in Marketing Research. *Intelligent Information Management*, *03*(05), 167–174. https://doi.org/10.4236/iim.2011.35020

Mauri, D. (2012). Temporal Snapshot Fact Table. Retrieved July 15, 2018, from https://sqlbits.com/Sessions/Event10/Temporal_Snapshot_Fact_Table

Microsoft. (2013). Microsoft SQL Server Data Tools. Retrieved from https://msdn.microsoft.com/en-us/library/mt204009.aspx

Ng, R., Arocena, P., Barbosa, D., Carenini, G., Gomes, L., Jou, S., … Yu, E. (2013). Perspectives on Business Intelligence. *Synthesis Lectures on Data Management*, *5*(1), 7–104. https://doi.org/10.2200/S00491ED1V01Y201303DTM034

Oracle Corporation. (2010). Oracle Advanced Pricing User's Guide. Retrieved June 22, 2018, from https://docs.oracle.com/cd/E18727_01/doc.121/e13427/T328362T328365.htm

Oracle Corporation. (2015). Oracle Advanced Pricing. Retrieved June 15, 2018, from http://www.oracle.com/us/products/applications/057142.pdf

Paris, E. H., & Washington, S. L. (2018). *Skill Shift Automation and the Future of the Workforce*. *McKinsey*.

Qlik. (2019). Qlik Sense. Retrieved from http://www.qlik.com/products/qlik-sense

Qlik. (2020). The Associative Difference. Retrieved March 2, 2020, from https://www.qlik.com/us/-/media/files/resource-library/global-us/direct/datasheets/ds-the-associative-difference-en.pdf

R, Kimball;R, M. (2008). *The Data Warehouse Lifecycle Toolkit*. *Architecture*.

Shahin, A., & Mahbod, M. A. (2007). Prioritization of key performance indicators: An integration of analytical hierarchy process and goal setting. *International Journal of Productivity and Performance Management*, *56*(3), 226–240. https://doi.org/10.1108/17410400710731437

Turban, E., Sharda, R., & Aronson, J. (2011). Business intelligence: a managerial approach. In *Tamu-Commerce.Edu* (pp. 3–27). https://doi.org/10.1109/HICSS.2012.138

W.H. Inmon, Derek Strauss, G. N. (2008). *DW 2.0: The Architecture for the Next Generation of Data Warehousing (Morgan Kaufman Series in Data Management Systems) (): : Books*. (Morgan Kaufmann, Ed.) (1st Editio). Retrieved from http://www.amazon.com/2-0-Architecture-Generation-Warehousing-Management/dp/0123743192

Wixom, B., & Watson, H. (2010). The BI-Based Organization. *International Journal of Business Intelligence Research*, *1*(1), 13–28. https://doi.org/10.4018/jbir.2010071702

7. ATTACHMENTS

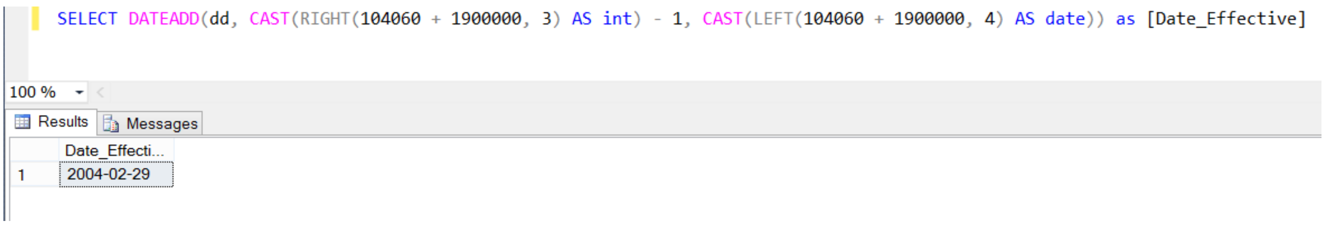
7.1. Date Conversion From JDE Julian Date to Gregorian Date

JDE Julian Date	Gregorian Date
Format: CYYDDD **C stands for century** **YY stands for year** **DDD stands for day of that year**	Format: **YYYY-MM-DD** **YYYY** stands for Year **MM** stands for Month **DD** stands for Day of the Month

Table 21 - Date Conversion

Validations:

Taking the SQL expression used in the SELECT statement for the column Date Effective, for example, in chapter 3.3.2.1 to convert the JDE Julian Date to Gregorian Date and hardcoding it with the number 104060, which is the 60th day of the leap year 2004, the following result is obtained:

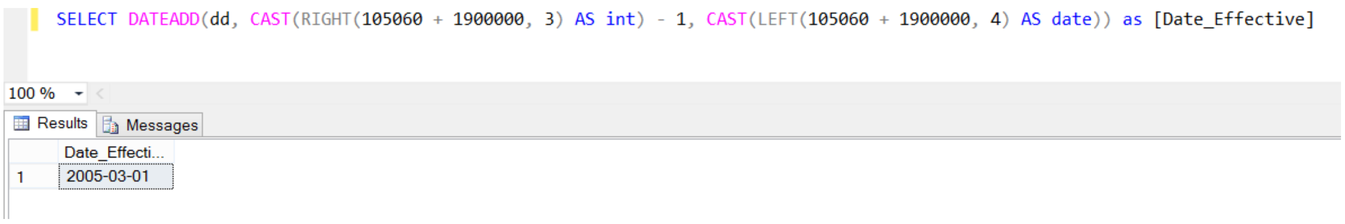


```
SELECT DATEADD(dd, CAST(RIGHT(104060 + 1900000, 3) AS int) - 1, CAST(LEFT(104060 + 1900000, 4) AS date)) as [Date_Effective]
```

```
   Date_Effecti...
1  2004-02-29
```

Figure 46 - Date Conversion - Leap Day Results

By using the same expression, but for the 60th day of the non-leap year 2005, the result is expected to be 2005-03-01, which is verified when executing the query:



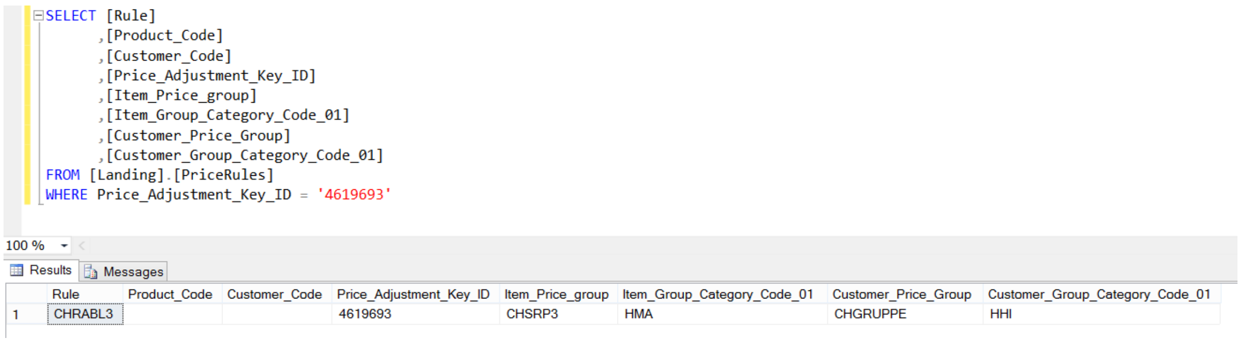
```
SELECT DATEADD(dd, CAST(RIGHT(105060 + 1900000, 3) AS int) - 1, CAST(LEFT(105060 + 1900000, 4) AS date)) as [Date_Effective]
```

```
   Date_Effecti...
1  2005-03-01
```

Figure 47 - Date Conversion - Non-leap Day Results

7.2. Cartesian product for blank Customer Codes and Product Codes

To validate that query in Figure 16 from subchapter 3.3.2.1 is doing the cartesian product when both Customer_Code and Product_Code are empty, the following example for the advanced pricing agreement with the Price_Adjustment_Key_ID 4619693 is analyzed:



```sql
SELECT [Rule]
      ,[Product_Code]
      ,[Customer_Code]
      ,[Price_Adjustment_Key_ID]
      ,[Item_Price_group]
      ,[Item_Group_Category_Code_01]
      ,[Customer_Price_Group]
      ,[Customer_Group_Category_Code_01]
FROM [Landing].[PriceRules]
WHERE Price_Adjustment_Key_ID = '4619693'
```

100 %

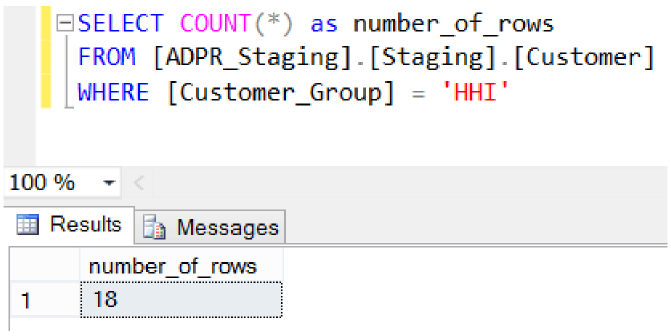
Results | Messages

	Rule	Product_Code	Customer_Code	Price_Adjustment_Key_ID	Item_Price_group	Item_Group_Category_Code_01	Customer_Price_Group	Customer_Group_Category_Code_01
1	CHRABL3			4619693	CHSRP3	HMA	CHGRUPPE	HHI

Figure 48 - Price_Adjustment_Key_ID Example

By querying the landing table PriceRules for this advanced pricing agreement, it's observed that both codes are empty.

Querying the Customer staging table for the customer group HHI, the number of customer codes is equal to 18, since every row in this table represents a different customer code.



```sql
SELECT COUNT(*) as number_of_rows
FROM [ADPR_Staging].[Staging].[Customer]
WHERE [Customer_Group] = 'HHI'
```

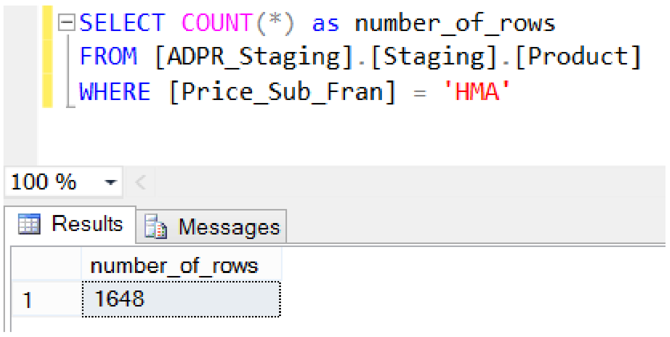
100 %

Results | Messages

	number_of_rows
1	18

Figure 49 - Number of Customer Codes for Customer Group HHI

Querying the Product staging table for the appropriate product group, there are 1648 different product codes.



```sql
SELECT COUNT(*) as number_of_rows
FROM [ADPR_Staging].[Staging].[Product]
WHERE [Price_Sub_Fran] = 'HMA'
```

100 %

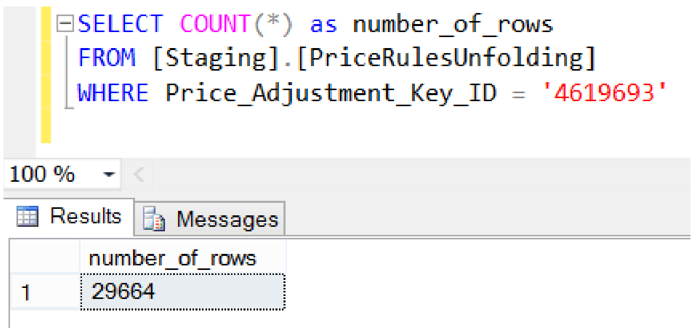
Results | Messages

	number_of_rows
1	1648

Figure 50 - Number of Product Codes for Price Sub Franchise HMA

Doing the cartesian product to ensure that there is a combination of all 18 customers by all 1648 products, the expected result is equal to 18 x 1648 = 29664.

This is verified after querying the table Staging.PriceRulesUnfolding, which is loaded with the output of the transformation query in Figure 16 from chapter 3.3.2.1:



```
SELECT COUNT(*) as number_of_rows
FROM [Staging].[PriceRulesUnfolding]
WHERE Price_Adjustment_Key_ID = '4619693'
```

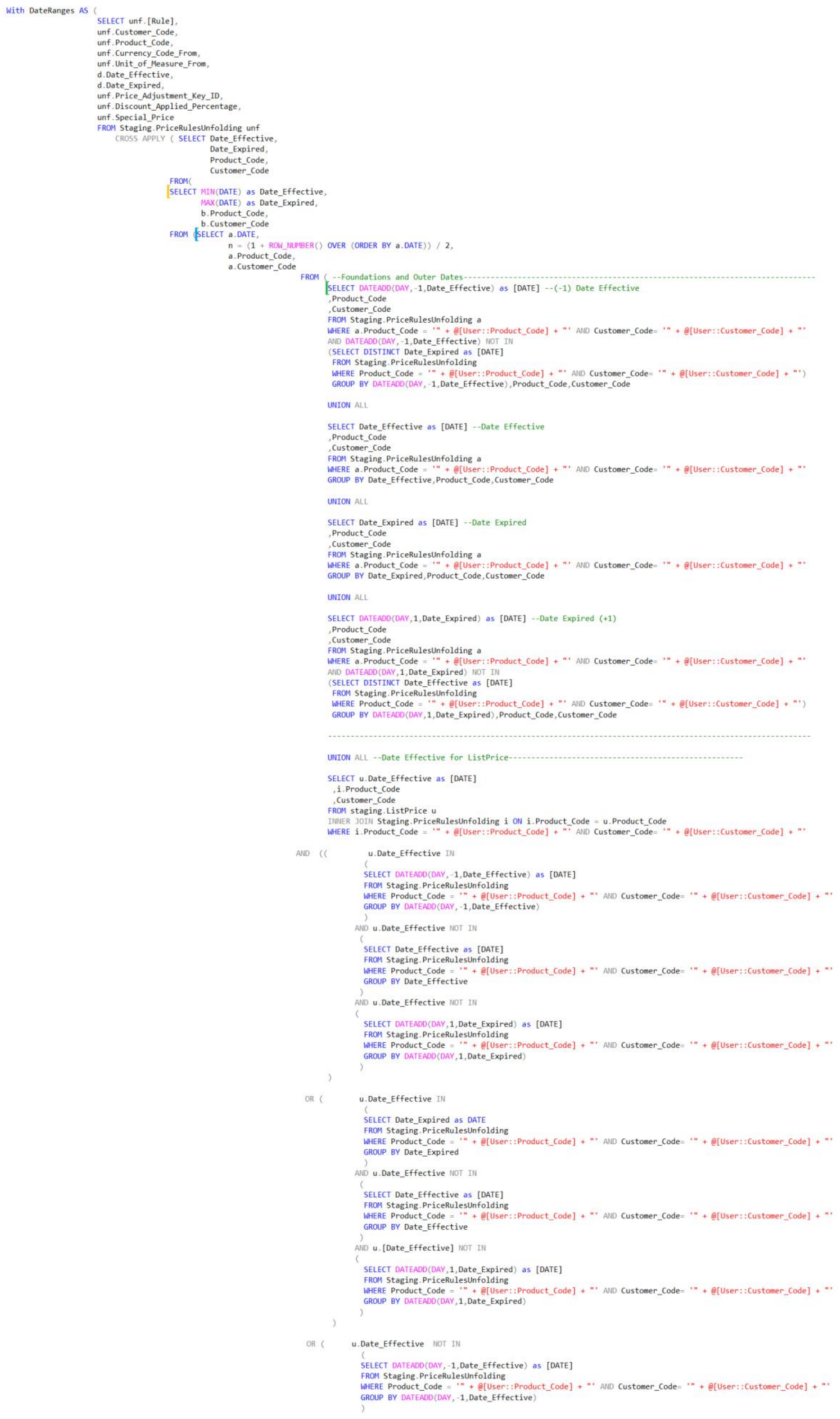
```
   number_of_rows
1  29664
```

Figure 51 - Cartesian Product Results

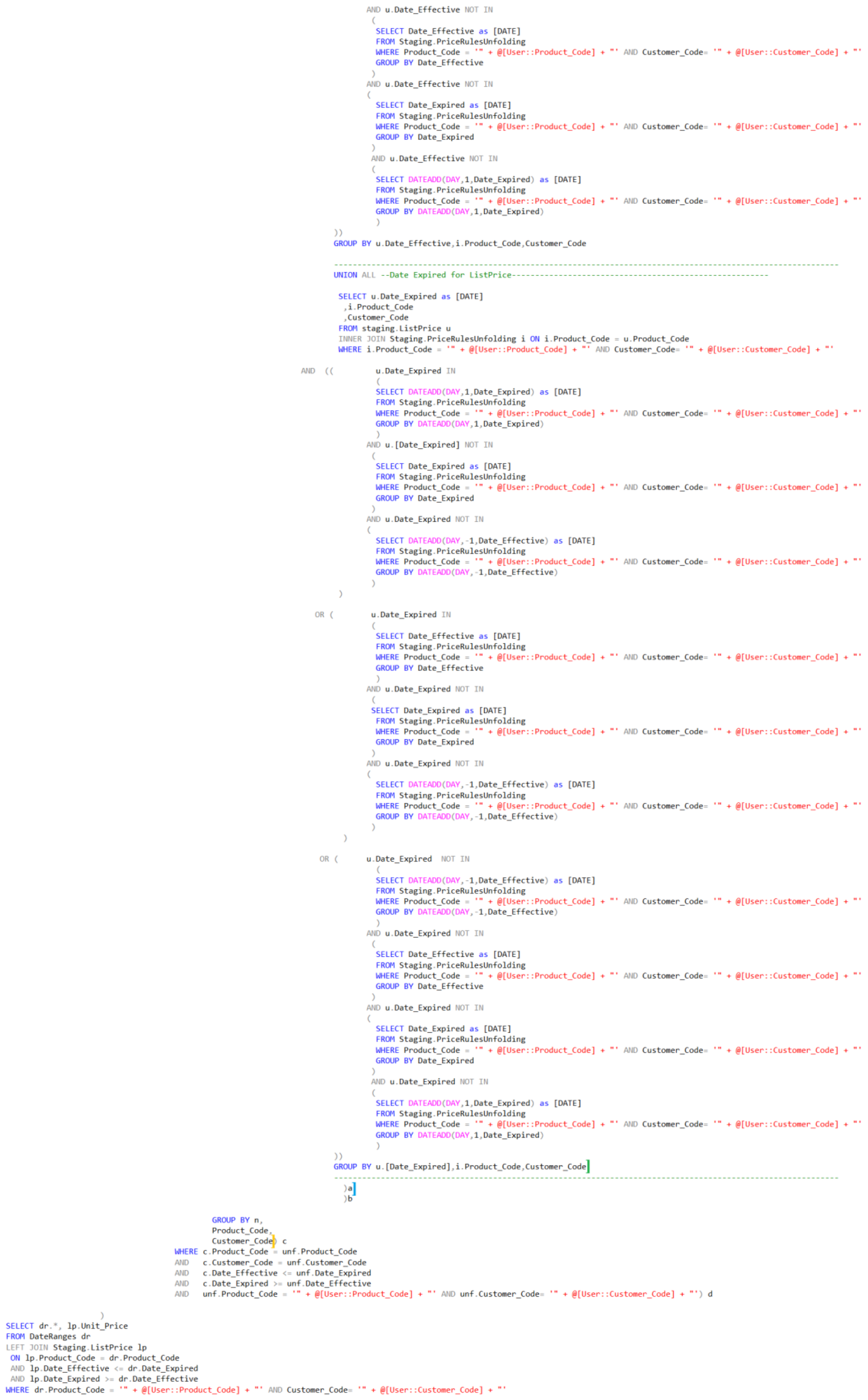
7.3. Date Ranges SSIS Package

7.3.1. Query - Date Ranges

Query used to separate the discounts into common date ranges for a specific customer code and product code taking into account the different discounts and list prices. This query also incorporates the date range aggregation rules:



```sql
With DateRanges AS (
                    SELECT unf.[Rule],
                    unf.Customer_Code,
                    unf.Product_Code,
                    unf.Currency_Code_From,
                    unf.Unit_of_Measure_From,
                    d.Date_Effective,
                    d.Date_Expired,
                    unf.Price_Adjustment_Key_ID,
                    unf.Discount_Applied_Percentage,
                    unf.Special_Price
                    FROM Staging.PriceRulesUnfolding unf
                        CROSS APPLY ( SELECT Date_Effective,
                                             Date_Expired,
                                             Product_Code,
                                             Customer_Code
                                FROM(
                                SELECT MIN(DATE) as Date_Effective,
                                       MAX(DATE) as Date_Expired,
                                       b.Product_Code,
                                       b.Customer_Code
                                FROM (SELECT a.DATE,
                                             n = (1 + ROW_NUMBER() OVER (ORDER BY a.DATE)) / 2,
                                             a.Product_Code,
                                             a.Customer_Code
                                                             FROM ( --Foundations and Outer Dates---------------------------------------------------------------------------------
                                                                   SELECT DATEADD(DAY,-1,Date_Effective) as [DATE] --(-1) Date Effective
                                                                   ,Product_Code
                                                                   ,Customer_Code
                                                                   FROM Staging.PriceRulesUnfolding a
                                                                   WHERE a.Product_Code = '" + @[User::Product_Code] + "' AND Customer_Code= '" + @[User::Customer_Code] + "'
                                                                   AND DATEADD(DAY,-1,Date_Effective) NOT IN
                                                                   (SELECT DISTINCT Date_Expired as [DATE]
                                                                    FROM Staging.PriceRulesUnfolding
                                                                    WHERE Product_Code = '" + @[User::Product_Code] + "' AND Customer_Code= '" + @[User::Customer_Code] + "')
                                                                    GROUP BY DATEADD(DAY,-1,Date_Effective),Product_Code,Customer_Code

                                                                   UNION ALL

                                                                   SELECT Date_Effective as [DATE] --Date Effective
                                                                   ,Product_Code
                                                                   ,Customer_Code
                                                                   FROM Staging.PriceRulesUnfolding a
                                                                   WHERE a.Product_Code = '" + @[User::Product_Code] + "' AND Customer_Code= '" + @[User::Customer_Code] + "'
                                                                   GROUP BY Date_Effective,Product_Code,Customer_Code

                                                                   UNION ALL

                                                                   SELECT Date_Expired as [DATE] --Date Expired
                                                                   ,Product_Code
                                                                   ,Customer_Code
                                                                   FROM Staging.PriceRulesUnfolding a
                                                                   WHERE a.Product_Code = '" + @[User::Product_Code] + "' AND Customer_Code= '" + @[User::Customer_Code] + "'
                                                                   GROUP BY Date_Expired,Product_Code,Customer_Code

                                                                   UNION ALL

                                                                   SELECT DATEADD(DAY,1,Date_Expired) as [DATE] --Date Expired (+1)
                                                                   ,Product_Code
                                                                   ,Customer_Code
                                                                   FROM Staging.PriceRulesUnfolding a
                                                                   WHERE a.Product_Code = '" + @[User::Product_Code] + "' AND Customer_Code= '" + @[User::Customer_Code] + "'
                                                                   AND DATEADD(DAY,1,Date_Expired) NOT IN
                                                                   (SELECT DISTINCT Date_Effective as [DATE]
                                                                    FROM Staging.PriceRulesUnfolding
                                                                    WHERE Product_Code = '" + @[User::Product_Code] + "' AND Customer_Code= '" + @[User::Customer_Code] + "')
                                                                    GROUP BY DATEADD(DAY,1,Date_Expired),Product_Code,Customer_Code

                                                                   --------------------------------------------------------------------------------------------------------------

                                                                   UNION ALL --Date Effective for ListPrice--------------------------------------------------

                                                                   SELECT u.Date_Effective as [DATE]
                                                                    ,i.Product_Code
                                                                    ,Customer_Code
                                                                   FROM staging.ListPrice u
                                                                   INNER JOIN Staging.PriceRulesUnfolding i ON i.Product_Code = u.Product_Code
                                                                   WHERE i.Product_Code = '" + @[User::Product_Code] + "' AND Customer_Code= '" + @[User::Customer_Code] + "'

                                                                AND  ((          u.Date_Effective IN
                                                                                 (
                                                                                SELECT DATEADD(DAY,-1,Date_Effective) as [DATE]
                                                                                FROM Staging.PriceRulesUnfolding
                                                                                WHERE Product_Code = '" + @[User::Product_Code] + "' AND Customer_Code= '" + @[User::Customer_Code] + "'
                                                                                GROUP BY DATEADD(DAY,-1,Date_Effective)
                                                                                 )
                                                                              AND u.Date_Effective NOT IN
                                                                               (
                                                                                SELECT Date_Effective as [DATE]
                                                                                FROM Staging.PriceRulesUnfolding
                                                                                WHERE Product_Code = '" + @[User::Product_Code] + "' AND Customer_Code= '" + @[User::Customer_Code] + "'
                                                                                GROUP BY Date_Effective
                                                                               )
                                                                              AND u.Date_Effective NOT IN
                                                                              (
                                                                                SELECT DATEADD(DAY,1,Date_Expired) as [DATE]
                                                                                FROM Staging.PriceRulesUnfolding
                                                                                WHERE Product_Code = '" + @[User::Product_Code] + "' AND Customer_Code= '" + @[User::Customer_Code] + "'
                                                                                GROUP BY DATEADD(DAY,1,Date_Expired)
                                                                               )
                                                                   )

                                                                  OR (          u.Date_Effective IN
                                                                                 (
                                                                                SELECT Date_Expired as DATE
                                                                                FROM Staging.PriceRulesUnfolding
                                                                                WHERE Product_Code = '" + @[User::Product_Code] + "' AND Customer_Code= '" + @[User::Customer_Code] + "'
                                                                                GROUP BY Date_Expired
                                                                                 )
                                                                              AND u.Date_Effective NOT IN
                                                                               (
                                                                                SELECT Date_Effective as [DATE]
                                                                                FROM Staging.PriceRulesUnfolding
                                                                                WHERE Product_Code = '" + @[User::Product_Code] + "' AND Customer_Code= '" + @[User::Customer_Code] + "'
                                                                                GROUP BY Date_Effective
                                                                               )
                                                                              AND u.[Date_Effective] NOT IN
                                                                              (
                                                                                SELECT DATEADD(DAY,1,Date_Expired) as [DATE]
                                                                                FROM Staging.PriceRulesUnfolding
                                                                                WHERE Product_Code = '" + @[User::Product_Code] + "' AND Customer_Code= '" + @[User::Customer_Code] + "'
                                                                                GROUP BY DATEADD(DAY,1,Date_Expired)
                                                                               )
                                                                    )

                                                                  OR (        u.Date_Effective  NOT IN
                                                                               (
                                                                               SELECT DATEADD(DAY,-1,Date_Effective) as [DATE]
                                                                               FROM Staging.PriceRulesUnfolding
                                                                               WHERE Product_Code = '" + @[User::Product_Code] + "' AND Customer_Code= '" + @[User::Customer_Code] + "'
                                                                               GROUP BY DATEADD(DAY,-1,Date_Effective)
                                                                               )
```



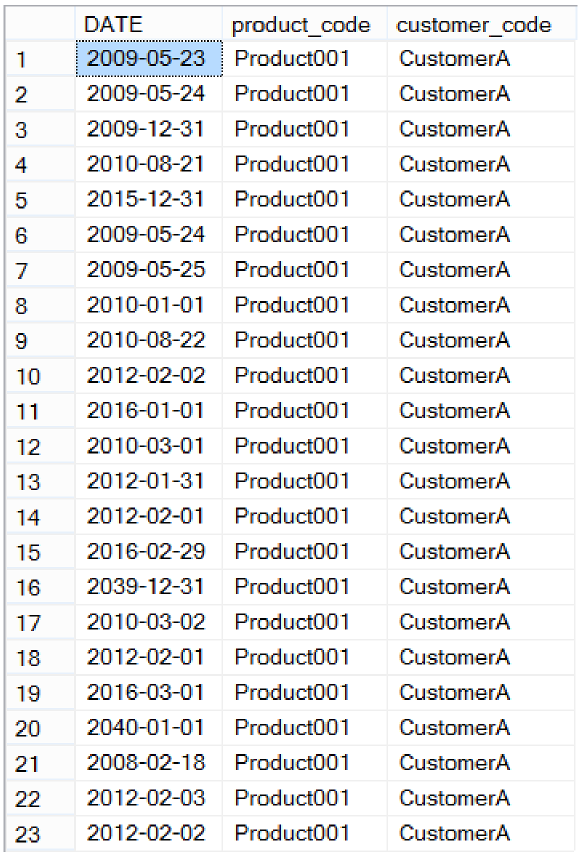
```sql
                                                                               AND u.Date_Effective NOT IN
                                                                                (
                                                                                 SELECT Date_Effective as [DATE]
                                                                                 FROM Staging.PriceRulesUnfolding
                                                                                 WHERE Product_Code = '" + @[User::Product_Code] + "' AND Customer_Code= '" + @[User::Customer_Code] + "'
                                                                                 GROUP BY Date_Effective
                                                                                )
                                                                               AND u.Date_Effective NOT IN
                                                                               (
                                                                                 SELECT Date_Expired as [DATE]
                                                                                 FROM Staging.PriceRulesUnfolding
                                                                                 WHERE Product_Code = '" + @[User::Product_Code] + "' AND Customer_Code= '" + @[User::Customer_Code] + "'
                                                                                 GROUP BY Date_Expired
                                                                                )
                                                                                AND u.Date_Effective NOT IN
                                                                                (
                                                                                 SELECT DATEADD(DAY,1,Date_Expired) as [DATE]
                                                                                 FROM Staging.PriceRulesUnfolding
                                                                                 WHERE Product_Code = '" + @[User::Product_Code] + "' AND Customer_Code= '" + @[User::Customer_Code] + "'
                                                                                 GROUP BY DATEADD(DAY,1,Date_Expired)
                                                                                 )
                                                                        ))
                                                                        GROUP BY u.Date_Effective,i.Product_Code,Customer_Code

                                                                        ------------------------------------------------------------------------------------------------------------
                                                                        UNION ALL --Date Expired for ListPrice----------------------------------------------------

                                                                         SELECT u.Date_Expired as [DATE]
                                                                          ,i.Product_Code
                                                                          ,Customer_Code
                                                                         FROM staging.ListPrice u
                                                                         INNER JOIN Staging.PriceRulesUnfolding i ON i.Product_Code = u.Product_Code
                                                                         WHERE i.Product_Code = '" + @[User::Product_Code] + "' AND Customer_Code= '" + @[User::Customer_Code] + "'

                                                                 AND  ((         u.Date_Expired IN
                                                                                 (
                                                                                 SELECT DATEADD(DAY,1,Date_Expired) as [DATE]
                                                                                 FROM Staging.PriceRulesUnfolding
                                                                                 WHERE Product_Code = '" + @[User::Product_Code] + "' AND Customer_Code= '" + @[User::Customer_Code] + "'
                                                                                 GROUP BY DATEADD(DAY,1,Date_Expired)
                                                                                 )
                                                                               AND u.[Date_Expired] NOT IN
                                                                                (
                                                                                 SELECT Date_Expired as [DATE]
                                                                                 FROM Staging.PriceRulesUnfolding
                                                                                 WHERE Product_Code = '" + @[User::Product_Code] + "' AND Customer_Code= '" + @[User::Customer_Code] + "'
                                                                                 GROUP BY Date_Expired
                                                                                )
                                                                               AND u.Date_Expired NOT IN
                                                                               (
                                                                                 SELECT DATEADD(DAY,-1,Date_Effective) as [DATE]
                                                                                 FROM Staging.PriceRulesUnfolding
                                                                                 WHERE Product_Code = '" + @[User::Product_Code] + "' AND Customer_Code= '" + @[User::Customer_Code] + "'
                                                                                 GROUP BY DATEADD(DAY,-1,Date_Effective)
                                                                                )
                                                                         )

                                                                    OR (         u.Date_Expired IN
                                                                                (
                                                                                 SELECT Date_Effective as [DATE]
                                                                                 FROM Staging.PriceRulesUnfolding
                                                                                 WHERE Product_Code = '" + @[User::Product_Code] + "' AND Customer_Code= '" + @[User::Customer_Code] + "'
                                                                                 GROUP BY Date_Effective
                                                                                 )
                                                                               AND u.Date_Expired NOT IN
                                                                                (
                                                                                SELECT Date_Expired as [DATE]
                                                                                 FROM Staging.PriceRulesUnfolding
                                                                                 WHERE Product_Code = '" + @[User::Product_Code] + "' AND Customer_Code= '" + @[User::Customer_Code] + "'
                                                                                 GROUP BY Date_Expired
                                                                                )
                                                                               AND u.Date_Expired NOT IN
                                                                               (
                                                                                 SELECT DATEADD(DAY,-1,Date_Effective) as [DATE]
                                                                                 FROM Staging.PriceRulesUnfolding
                                                                                 WHERE Product_Code = '" + @[User::Product_Code] + "' AND Customer_Code= '" + @[User::Customer_Code] + "'
                                                                                 GROUP BY DATEADD(DAY,-1,Date_Effective)
                                                                                )
                                                                          )

                                                                     OR (       u.Date_Expired  NOT IN
                                                                                 (
                                                                                 SELECT DATEADD(DAY,-1,Date_Effective) as [DATE]
                                                                                 FROM Staging.PriceRulesUnfolding
                                                                                 WHERE Product_Code = '" + @[User::Product_Code] + "' AND Customer_Code= '" + @[User::Customer_Code] + "'
                                                                                 GROUP BY DATEADD(DAY,-1,Date_Effective)
                                                                                 )
                                                                               AND u.Date_Expired NOT IN
                                                                                (
                                                                                 SELECT Date_Effective as [DATE]
                                                                                 FROM Staging.PriceRulesUnfolding
                                                                                 WHERE Product_Code = '" + @[User::Product_Code] + "' AND Customer_Code= '" + @[User::Customer_Code] + "'
                                                                                 GROUP BY Date_Effective
                                                                                )
                                                                               AND u.Date_Expired NOT IN
                                                                               (
                                                                                 SELECT Date_Expired as [DATE]
                                                                                 FROM Staging.PriceRulesUnfolding
                                                                                 WHERE Product_Code = '" + @[User::Product_Code] + "' AND Customer_Code= '" + @[User::Customer_Code] + "'
                                                                                 GROUP BY Date_Expired
                                                                                )
                                                                                AND u.Date_Expired NOT IN
                                                                                (
                                                                                 SELECT DATEADD(DAY,1,Date_Expired) as [DATE]
                                                                                 FROM Staging.PriceRulesUnfolding
                                                                                 WHERE Product_Code = '" + @[User::Product_Code] + "' AND Customer_Code= '" + @[User::Customer_Code] + "'
                                                                                 GROUP BY DATEADD(DAY,1,Date_Expired)
                                                                                 )
                                                                        ))
                                                                        GROUP BY u.[Date_Expired],i.Product_Code,Customer_Code
                                                                        ------------------------------------------------------------------------------------------------------------
                                                                          )a
                                                                          )b

                                             GROUP BY n,
                                             Product_Code,
                                             Customer_Code) c
                                     WHERE c.Product_Code = unf.Product_Code
                                     AND   c.Customer_Code = unf.Customer_Code
                                     AND   c.Date_Effective <= unf.Date_Expired
                                     AND   c.Date_Expired >= unf.Date_Effective
                                     AND   unf.Product_Code = '" + @[User::Product_Code] + "' AND unf.Customer_Code= '" + @[User::Customer_Code] + "') d

                     )
SELECT dr.*, lp.Unit_Price
FROM DateRanges dr
LEFT JOIN Staging.ListPrice lp
 ON lp.Product_Code = dr.Product_Code
 AND lp.Date_Effective <= dr.Date_Expired
 AND lp.Date_Expired >= dr.Date_Effective
WHERE dr.Product_Code = '" + @[User::Product_Code] + "' AND Customer_Code= '" + @[User::Customer_Code] + "'
```

Figure 52 - Query Date Ranges

The query in Figure 52 is used as a variable for every combination of product code and customer code. For illustration purposes, the query holds 3 colored square brackets and is hardcoded for CustomerA and Product001.

Looking at the inner subquery, which can be found within the green square brackets, we have the pool of dates. Again, using Figure 23 as the reference, all the valid dates that are generated from the aggregation rules for CustomerA and Product001 are listed individually here.

The subquery can be divided into 3 segments. The first segment lists the distinct foundations and outer dates that passed the conditions of Table 14. The same conditions are applied to the second segment, which lists all the Date Effective of the List Price, and the third segment lists all the valid Date Expired of the List Price. In the end by executing the subquery, we get the following result:



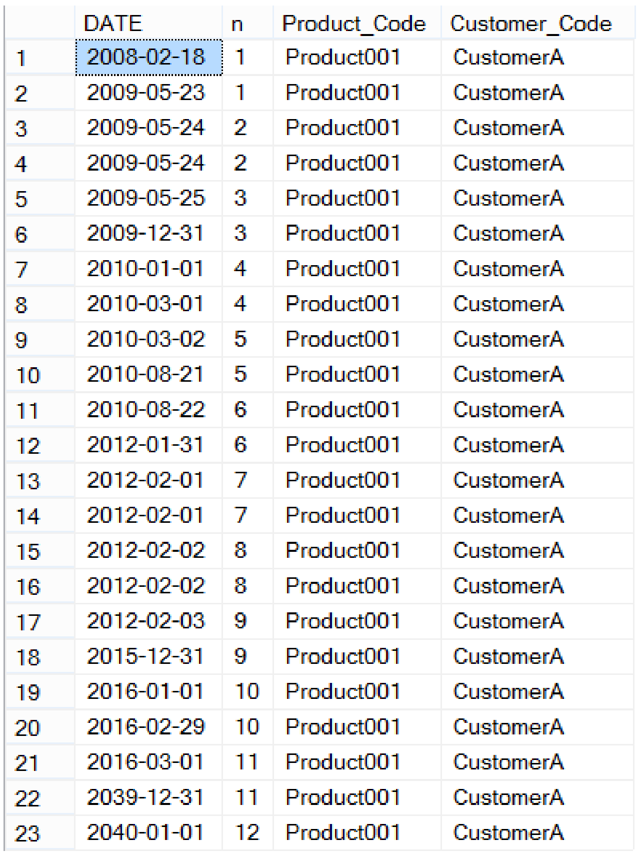
	DATE	product_code	customer_code
1	2009-05-23	Product001	CustomerA
2	2009-05-24	Product001	CustomerA
3	2009-12-31	Product001	CustomerA
4	2010-08-21	Product001	CustomerA
5	2015-12-31	Product001	CustomerA
6	2009-05-24	Product001	CustomerA
7	2009-05-25	Product001	CustomerA
8	2010-01-01	Product001	CustomerA
9	2010-08-22	Product001	CustomerA
10	2012-02-02	Product001	CustomerA
11	2016-01-01	Product001	CustomerA
12	2010-03-01	Product001	CustomerA
13	2012-01-31	Product001	CustomerA
14	2012-02-01	Product001	CustomerA
15	2016-02-29	Product001	CustomerA
16	2039-12-31	Product001	CustomerA
17	2010-03-02	Product001	CustomerA
18	2012-02-01	Product001	CustomerA
19	2016-03-01	Product001	CustomerA
20	2040-01-01	Product001	CustomerA
21	2008-02-18	Product001	CustomerA
22	2012-02-03	Product001	CustomerA
23	2012-02-02	Product001	CustomerA

Table 22 - Subquery Pool of Dates Result [Green Square Brackets]

This result is comprised of all the dates that are highlighted in blue in Figure 23.

The second step to understand the query is to look at the subquery within the Blue Square Brackets, which uses a ROW_NUMBER in a column named "n" to easily group the dates in pairs and order them in an ascended manner.

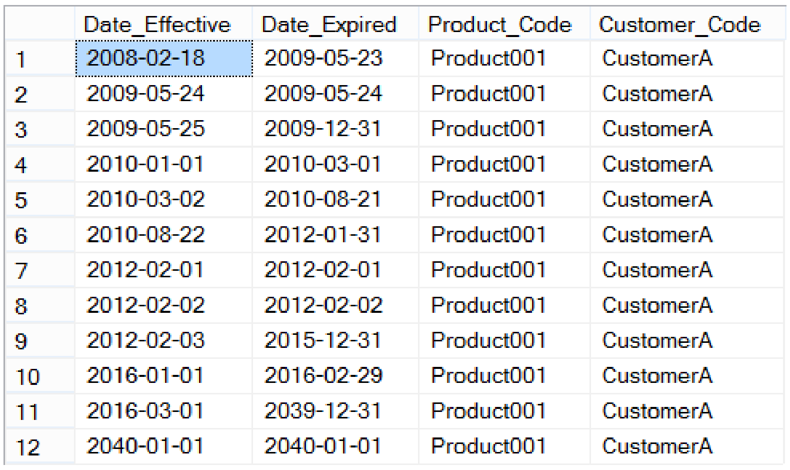
The outcome of this blue square bracket subquery is shown in Table 23:



	DATE	n	Product_Code	Customer_Code
1	2008-02-18	1	Product001	CustomerA
2	2009-05-23	1	Product001	CustomerA
3	2009-05-24	2	Product001	CustomerA
4	2009-05-24	2	Product001	CustomerA
5	2009-05-25	3	Product001	CustomerA
6	2009-12-31	3	Product001	CustomerA
7	2010-01-01	4	Product001	CustomerA
8	2010-03-01	4	Product001	CustomerA
9	2010-03-02	5	Product001	CustomerA
10	2010-08-21	5	Product001	CustomerA
11	2010-08-22	6	Product001	CustomerA
12	2012-01-31	6	Product001	CustomerA
13	2012-02-01	7	Product001	CustomerA
14	2012-02-01	7	Product001	CustomerA
15	2012-02-02	8	Product001	CustomerA
16	2012-02-02	8	Product001	CustomerA
17	2012-02-03	9	Product001	CustomerA
18	2015-12-31	9	Product001	CustomerA
19	2016-01-01	10	Product001	CustomerA
20	2016-02-29	10	Product001	CustomerA
21	2016-03-01	11	Product001	CustomerA
22	2039-12-31	11	Product001	CustomerA
23	2040-01-01	12	Product001	CustomerA

Table 23 - Pool Dates in Pairs Result [Blue Square Brackets]

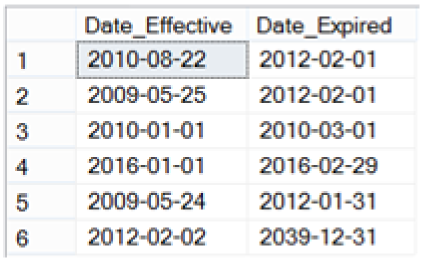
The next step is covered within the Orange Square Brackets. This transformation separates the minimum and maximum dates for every "n" into a date effective and date expired. In essence, it makes the previous sorting into clear Date Effective and Date Expired. The result is verified in Table 24:



	Date_Effective	Date_Expired	Product_Code	Customer_Code
1	2008-02-18	2009-05-23	Product001	CustomerA
2	2009-05-24	2009-05-24	Product001	CustomerA
3	2009-05-25	2009-12-31	Product001	CustomerA
4	2010-01-01	2010-03-01	Product001	CustomerA
5	2010-03-02	2010-08-21	Product001	CustomerA
6	2010-08-22	2012-01-31	Product001	CustomerA
7	2012-02-01	2012-02-01	Product001	CustomerA
8	2012-02-02	2012-02-02	Product001	CustomerA
9	2012-02-03	2015-12-31	Product001	CustomerA
10	2016-01-01	2016-02-29	Product001	CustomerA
11	2016-03-01	2039-12-31	Product001	CustomerA
12	2040-01-01	2040-01-01	Product001	CustomerA

Table 24 - Pool Dates in Date Effective and Date Expired [Orange Square Brackets]

For the next step, to obtain the additional columns that are needed and to obtain only the date ranges that have a discount associated, the whole CTE query can be executed. This essentially takes the previous query result and cross applies it with the Staging.PriceRulesUnfolding table where the dates are within the lowest and highest valid foundation dates, shown in Table 25.



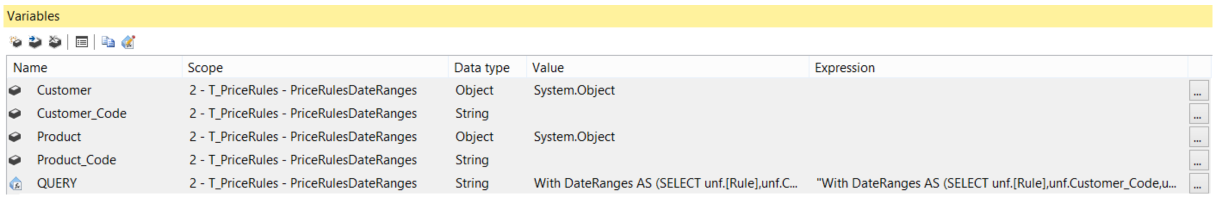
	Date_Effective	Date_Expired
1	2010-08-22	2012-02-01
2	2009-05-25	2012-02-01
3	2010-01-01	2010-03-01
4	2016-01-01	2016-02-29
5	2009-05-24	2012-01-31
6	2012-02-02	2039-12-31

Table 25 - Dates from Staging.PriceRulesUnfolding

The CTE will provide the results shown in the table 15, except for the List Price (still labeled Unit_Price as this stage) column. To get the list price, the CTE is then left joined with the Staging.Listprice table where the list price is active within the Date Effective and Date Expired already defined.

7.3.2. Date Ranges Control Flow Details

After observing Figure 25 with the overall picture of the Control Flow for the Date Ranges, the first step is to create the variables for the package.


Variables

Name	Scope	Data type	Value	Expression
Customer	2 - T_PriceRules - PriceRulesDateRanges	Object	System.Object	
Customer_Code	2 - T_PriceRules - PriceRulesDateRanges	String		
Product	2 - T_PriceRules - PriceRulesDateRanges	Object	System.Object	
Product_Code	2 - T_PriceRules - PriceRulesDateRanges	String		
QUERY	2 - T_PriceRules - PriceRulesDateRanges	String	With DateRanges AS (SELECT unf.[Rule],unf.C...	"With DateRanges AS (SELECT unf.[Rule],unf.Customer_Code,u...

Figure 53 - Date Ranges - SSIS Variables

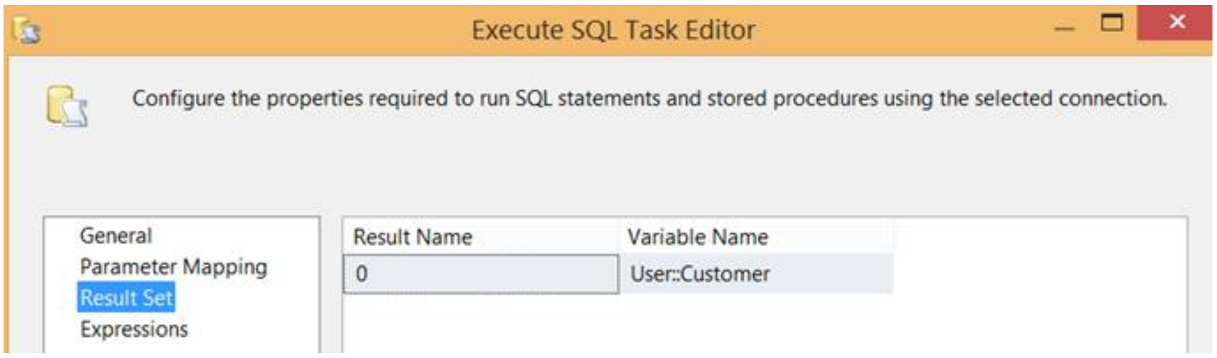
The Customer and Product system objects are variables creates to store the result set that comes from the query that is being executed in the Execute SQL Task components named “Get all Customers” and “Get all Products”.

The Customer_Code and Product_Code are variables of the string data type that hold the value of the respective customer_code and product_code columns when they are being cycled through in the for each loop containers.

The variable “QUERY” holds the query in Figure 52.

The SSIS Package starts with a component to truncate the Staging.PriceRulesDateRanges table, which is the standard process for staging tables.

The second component, which has the SQL statement visible as a note in Figure 25, lists all the distinct customer_codes that exist in the Staging.PriceRulesUnfolding table. This component uses the variable “Customer” to store the result set, as shown in Figure 54:

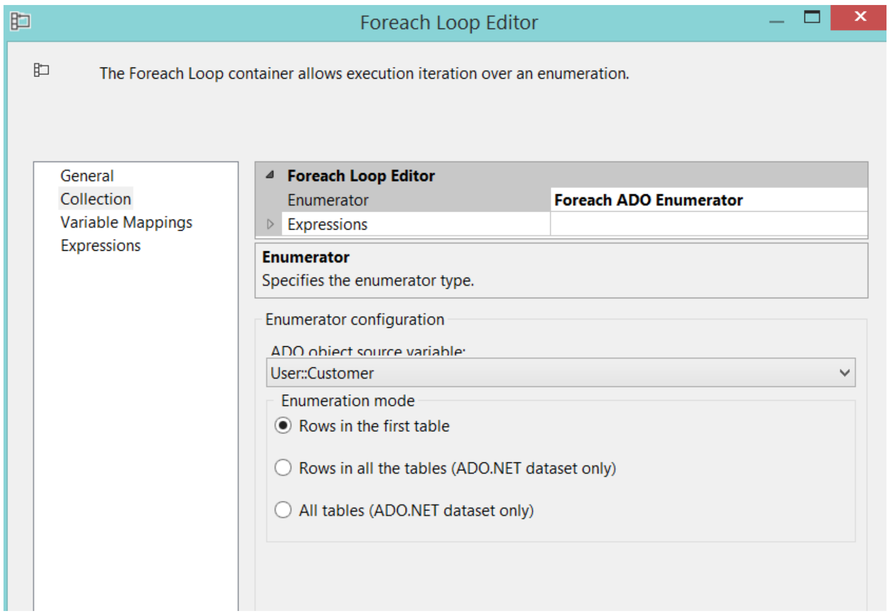


Configure the properties required to run SQL statements and stored procedures using the selected connection.

Result Name	Variable Name
0	User::Customer

Figure 54 - Date Ranges - Execute SQL Task Result Set

The component Foreach Loop Container Customer uses the Foreach Ado Enumerator to loop the Customer variable in the Collection tab and then map it to the Customer_Code variable in the Variable Mappings tab.


Foreach Loop Editor

The Foreach Loop container allows execution iteration over an enumeration.

Foreach Loop Editor	
Enumerator	Foreach ADO Enumerator
Expressions	

Enumerator
Specifies the enumerator type.

Enumerator configuration

ADO object source variable: User::Customer

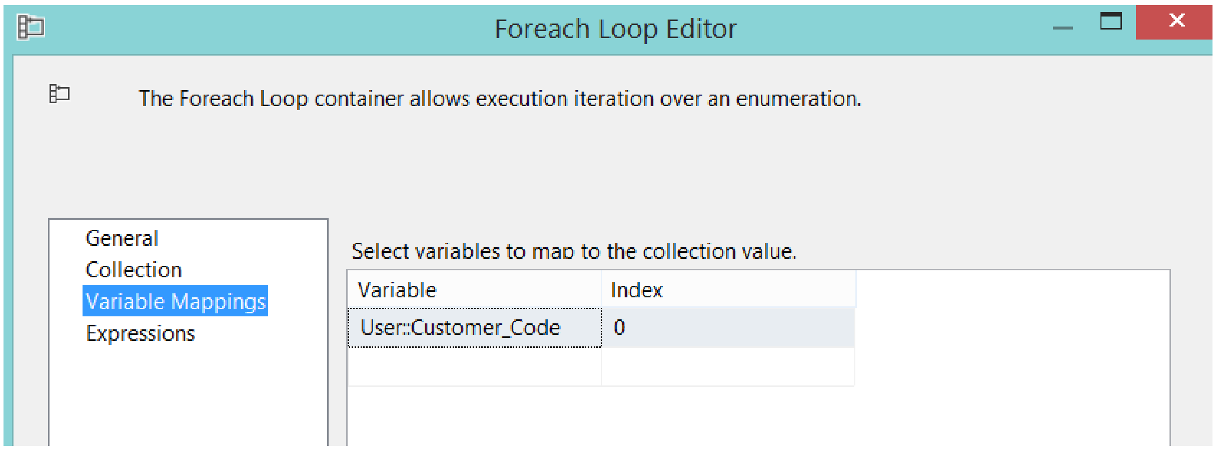
Enumeration mode

● Rows in the first table

○ Rows in all the tables (ADO.NET dataset only)

○ All tables (ADO.NET dataset only)

Figure 55 - Date Ranges - Foreach Loop Container Customer - Collection


Foreach Loop Editor

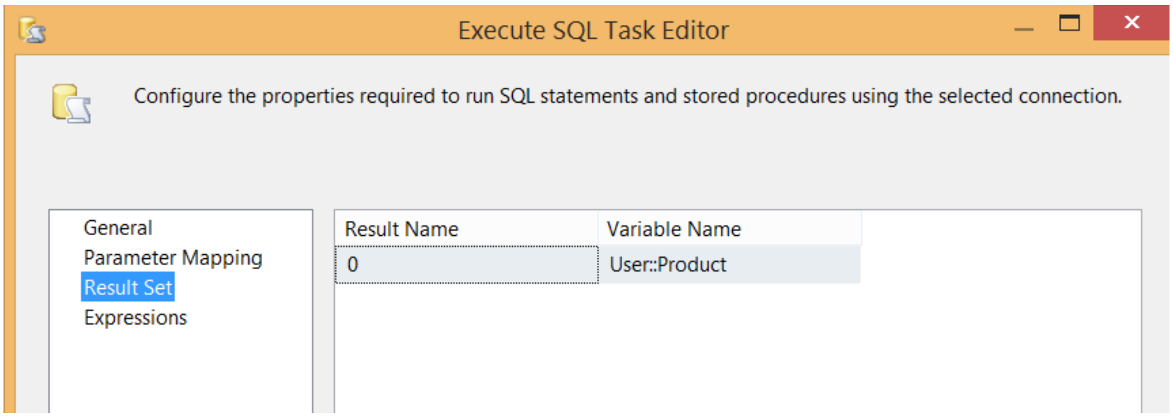
The Foreach Loop container allows execution iteration over an enumeration.

Select variables to map to the collection value.

Variable	Index
User::Customer_Code	0

Figure 56 - Date Ranges - Foreach Loop Container Customer - Variable Mappings

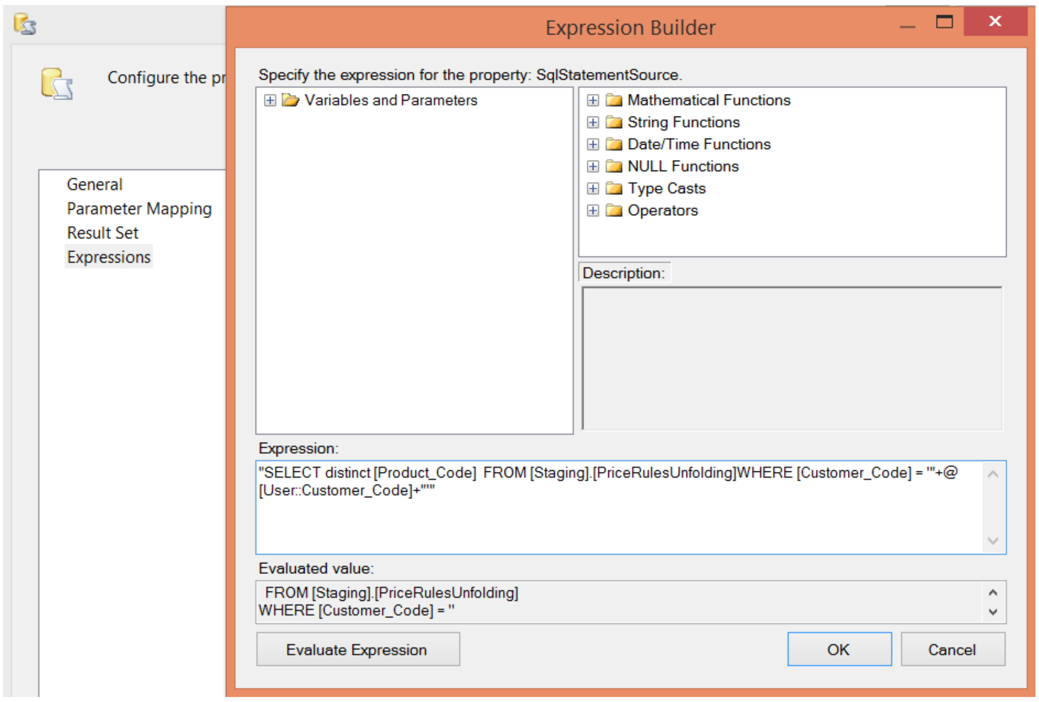
The Execute SQL Task component labeled “Get all Products” has the purpose of matching every product to the existing customers being looped. For this combination to take effect, there’s an expression that is introduced in the Expressions tab, where the property SQLStatementSource holds a query to get all the distinct product codes and stating that the Customer_Code must be equal to the Customer_Code variable, as shown in Figure 58.



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

Result Name	Variable Name
0	User::Product

Figure 57 - Date Ranges - Execute SQL Task - Get All Products - Result Set



Specify the expression for the property: SqlStatementSource.

Expression:

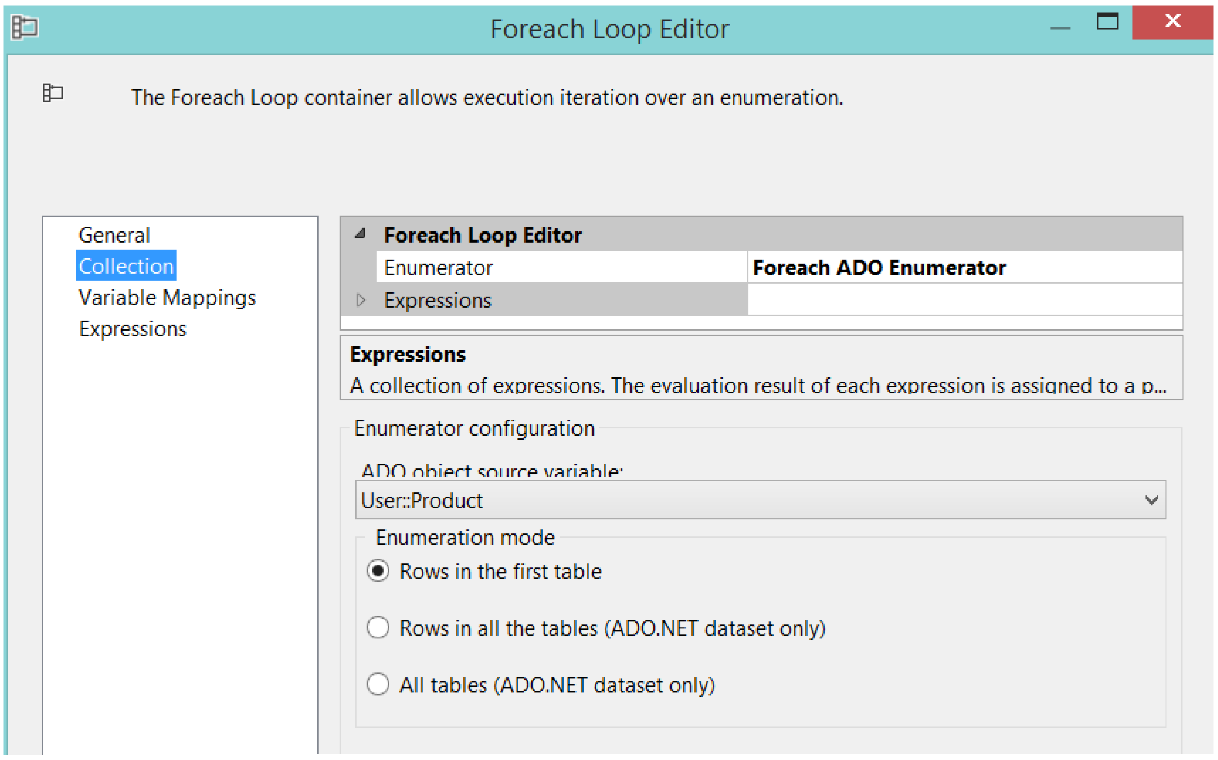
```
"SELECT distinct [Product_Code]  FROM [Staging].[PriceRulesUnfolding]WHERE [Customer_Code] = '"+@
[User::Customer_Code]+"'"
```

Evaluated value:

```
 FROM [Staging].[PriceRulesUnfolding]
WHERE [Customer_Code] = ''
```

Figure 58 - Date Ranges - Execute SQL Task Get All Products - Expression Builder

The second Foreach Loop Container seeks to put in place the process of looping all the products for every customer. The configurations are the same as the Customer ForEach Loop Container, but the main difference is that the Product one is contained within the Customer one.



The Foreach Loop container allows execution iteration over an enumeration.

Foreach Loop Editor

Enumerator	**Foreach ADO Enumerator**
Expressions	

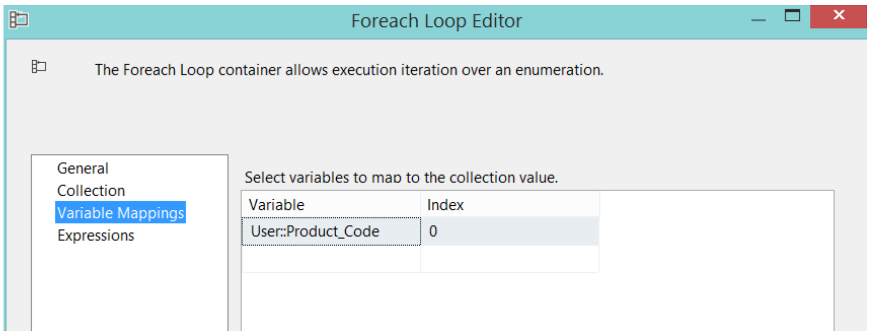
Expressions
A collection of expressions. The evaluation result of each expression is assigned to a p...

Enumerator configuration

ADO object source variable:
User::Product

Enumeration mode
- ● Rows in the first table
- ○ Rows in all the tables (ADO.NET dataset only)
- ○ All tables (ADO.NET dataset only)

Figure 59 - Date Ranges - Foreach Loop Container Product - Collection



The Foreach Loop container allows execution iteration over an enumeration.

Select variables to map to the collection value.

Variable	Index
User::Product_Code	0

Figure 60 - Date Ranges - Foreach Loop Container Product - Variable Mappings

7.4. Date Ranges Aggregation Rules

Analysis and validations to demonstrate how the Date Range Aggregation Rules table was formulated.

For explanation purposes, the dates will be named as Foundation Discount Dates, Outer Discount Dates, and List Price Dates.

For the analysis illustrations, the column that is being analyzed is marked with a purple cell above. The date that is being analyzed is marked either in green or red, depending on whether it is being considered or not as a date to be inserted, respectively.

7.4.1. Foundation Discount Dates

The first step is to establish the rules for the Foundation Discount Dates. They consist of the distinct dates of Date Effective and Date Expired. For every combination of customer and product, even if there are multiple Date Effective for the same date, they will always be considered, but only once. The same is true for the Date Expired. Only one Date Expired will be considered for a particular date.

When to Insert Distinct Foundation Discount Dates		
Analysis nr	Discount	Action
1	Date Effective	INSERT
2	Date Expired	INSERT

Table 26 - When to Insert Distinct Foundation Discount Dates

7.4.2. Outer Discount Dates

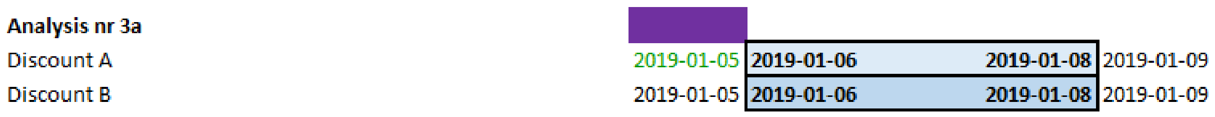
For the Outer Discount Dates, analysis 3 through 10 ensure that all scenarios for the Date Effective (-1) and Date Expired (+1) with another discount date are analyzed in order to be considered or not in the aggregation rules.

When to Insert Distinct Outer Discount Dates (Discount A)			
Analysis nr	Discount A	Discount B	Action
3	Date Effective (-1)	Date Effective (-1)	INSERT
4		Date Effective	INSERT
5		Date Expired	DON'T INSERT
6		Date Expired (+1)	INSERT
7	Date Expired (+1)	Date Effective (-1)	INSERT
8		Date Effective	DON'T INSERT
9		Date Expired	INSERT
10		Date Expired (+1)	INSERT

Table 27 - When to Insert Distinct Outer Discount Dates

7.4.2.1. Analysis 3

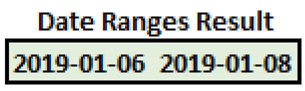
In this scenario, the Date Effective (-1) from discount A is analyzed in the event that there's another Date Effective (-1) occurring on the same date. This Date Effective (-1) will be considered once to correctly segregate the date ranges. Although in this context, since this outer Date Effective (-1) lies outside of the actual date ranges for discounts A and B, it will end up not being considered in the final solution, which only arranges the dates that actually have an advanced pricing agreement in place. Nonetheless, as part of the aggregation rules, it is considered.



Analysis nr 3a				
Discount A	2019-01-05	**2019-01-06**	**2019-01-08**	2019-01-09
Discount B	2019-01-05	**2019-01-06**	**2019-01-08**	2019-01-09

Figure 61 - Date Ranges Aggregation Rules - Analysis 3a

The final solution with the date ranges that contain a discount is:

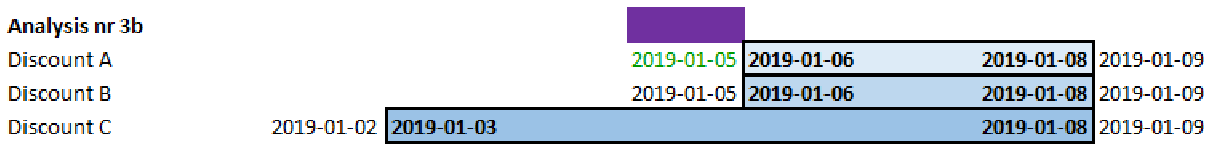


Date Ranges Result	
2019-01-06	**2019-01-08**

Figure 62 - Date Ranges Aggregation Rules - Analysis 3a Results

From the distinct Date Effective (2019-01-06) to the distinct Date Expired (2019-01-08) of discounts A and B, both discounts have a common date range.

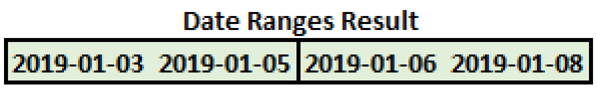
Considering the previous example, but adding a third discount C, it becomes clearer that this scenario must consider the Date Effective (-1).



Analysis nr 3b					
Discount A			2019-01-05	**2019-01-06**	**2019-01-08**	2019-01-09
Discount B			2019-01-05	**2019-01-06**	**2019-01-08**	2019-01-09
Discount C	2019-01-02	**2019-01-03**			**2019-01-08**	2019-01-09

Figure 63 - Date Ranges Aggregation Rules - Analysis 3b

The final solution now also considers the Date Effective (-1) because it is used to correctly separate the common discounts into the following date ranges:



Date Ranges Result			
2019-01-03	2019-01-05	2019-01-06	2019-01-08

Figure 64 - Date Ranges Aggregation Rules - Analysis 3b Results

From the Date Effective (2019-01-03) of discount C to the distinct Date Effective (-1) (2019-01-05) of both discounts A and B, only the discount C is valid. From the distinct Date Effective (2019-01-06) from discounts A and B to the distinct Date Expired (2019-01-08) of discounts A, B and C, all three discounts share a common date range.

7.4.2.2. Analysis 4

In this scenario, the Date Effective (-1) from discount A is analyzed in the event that there's a Date Effective occurring on the same date. The Date Effective (-1) will be considered. This allows for a correct segregation of the dates.

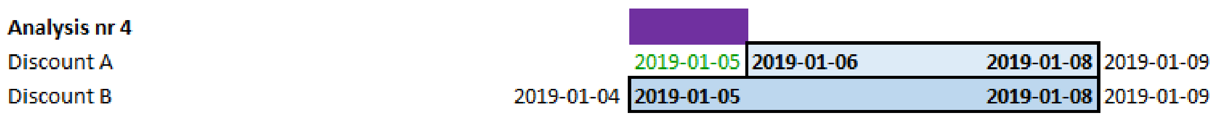
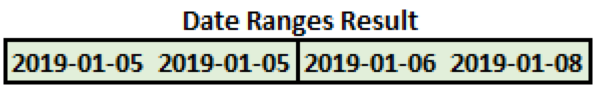


Figure 65 - Date Ranges Aggregation Rules - Analysis 4

The final solution with the date ranges that contain a discount is:



Date Ranges Result			
2019-01-05	2019-01-05	2019-01-06	2019-01-08

Figure 66 - Date Ranges Aggregation Rules - Analysis 4 Results

From the Date Effective (2019-01-05) of discount B to the Date Effective (-1) (2019-01-05) of discount A, only the discount B is valid. From the Date Effective (2019-01-06) of discount A to the distinct Date Expired (2019-01-08) of discounts A and B, both discounts share a common date range.

7.4.2.3. Analysis 5

In this scenario, the Date Effective (-1) from discount A is analyzed in the event that there's a Date Expired occurring on the same date. The Date Effective (-1) will not be considered, as the dates are already correctly divided.

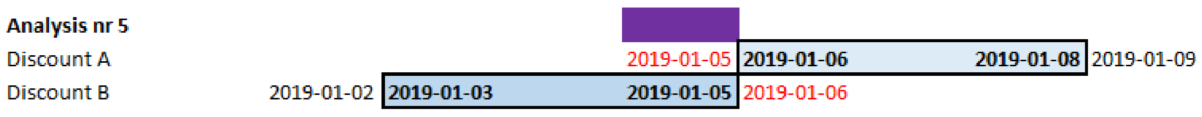


Figure 67 - Date Ranges Aggregation Rules - Analysis 5

The final solution with the date ranges that contain a discount is:

Date Ranges Result

2019-01-03 2019-01-05	2019-01-06 2019-01-08

Figure 68 - Date Ranges Aggregation Rules - Analysis 5 Results

From the Date Effective (2019-01-03) to the Date Expired (2019-01-05) of discount B, only the discount B is valid. From the Date Effective (2019-01-06) to the Date Expired (2019-01-08) of discount A, only the discount A is valid.

7.4.2.4. Analysis 6

In this scenario, the Date Effective (-1) from discount A is analyzed in the event that there's a Date Expired (+1) occurring on the same date. The Date Effective (-1) will be considered. This allows for a correct segregation of the dates.

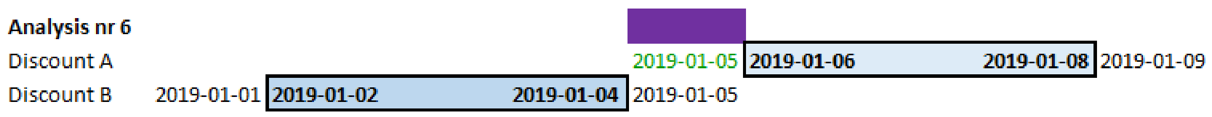


Figure 69 - Date Ranges Aggregation Rules - Analysis 6

The final solution with the date ranges that contain a discount is:

Date Ranges Result

2019-01-02 2019-01-04	2019-01-06 2019-01-08

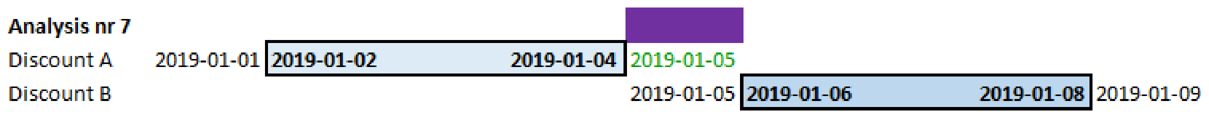
Figure 70 - Date Ranges Aggregation Rules - Analysis 6 Results

From the Date Effective (2019-01-02) to the Date Expired (2019-01-04) of discount B, only the discount B is valid. From the Date Effective (2019-01-06) to the Date Expired (2019-01-08) of discount A, only the discount A is valid.

Since the final solution only takes into account the date ranges with valid discounts, the date range 2019-01-05 to 2019-01-05 is not considered, but in the event that there was a third discount that existed covering this date, this date range consisting of this single day would be inserted.

7.4.2.5. Analysis 7

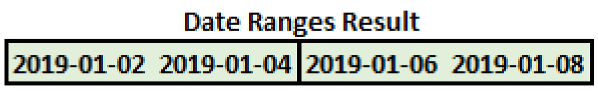
In this scenario, the Date Expired (+1) from discount A is analyzed in the event that there's a Date Effective (-1) occurring on the same date. The Date Expired (+1) will be considered. This allows for a correct segregation of the dates.



Analysis nr 7							
Discount A	2019-01-01	2019-01-02	2019-01-04	2019-01-05			
Discount B				2019-01-05	2019-01-06	2019-01-08	2019-01-09

Figure 71 - Date Ranges Aggregation Rules - Analysis 7

The final solution with the date ranges that contain a discount is:



Date Ranges Result			
2019-01-02	2019-01-04	2019-01-06	2019-01-08

Figure 72 - Date Ranges Aggregation Rules - Analysis 7 Results

From the Date Effective (2019-01-02) to the Date Expired (2019-01-04) of discount A, only the discount A is valid. From the Date Effective (2019-01-06) to the Date Expired (2019-01-08) of discount B, only the discount B is valid.

Since the final solution only takes into account the date ranges with valid discounts, the date range 2019-01-05 to 2019-01-05 is not considered, but in the event that there was a third discount that existed covering this date, this date range consisting of this single day would be inserted.

7.4.2.6. Analysis 8

In this scenario, the Date Expired (+1) from discount A is analyzed in the event that there's a Date Effective occurring on the same date. The Date Expired (+1) will not be considered, as the dates are already correctly divided.

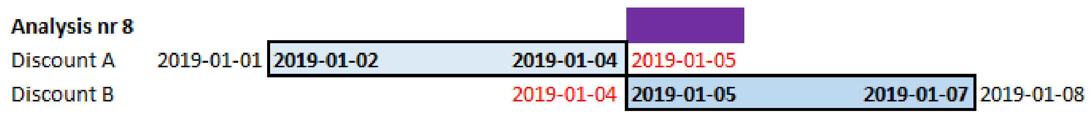
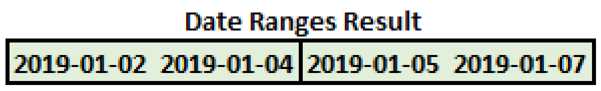


Figure 73 - Date Ranges Aggregation Rules - Analysis 8

The final solution with the date ranges that contain a discount is:


Date Ranges Result

2019-01-02 2019-01-04	2019-01-05 2019-01-07

Figure 74 - Date Ranges Aggregation Rules - Analysis 8 Results

From the Date Effective (2019-01-02) to the Date Expired (2019-01-04) of discount A, only the discount A is valid. From the Date Effective (2019-01-05) to the Date Expired (2019-01-07) of discount B, only the discount B is valid.

7.4.2.7. Analysis 9

In this scenario, the Date Expired (+1) from discount A is analyzed in the event that there's a Date Expired occurring on the same date. The Date Expired (+1) will be considered. This allows for a correct segregation of the dates.

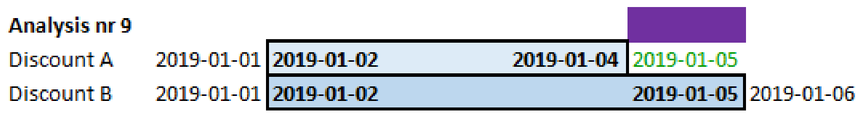
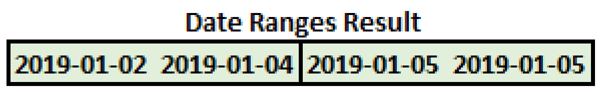


Figure 75 - Date Ranges Aggregation Rules - Analysis 9

The final solution with the date ranges that contain a discount is:


Date Ranges Result

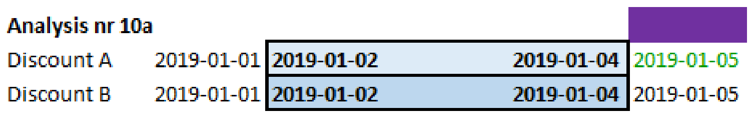
2019-01-02 2019-01-04	2019-01-05 2019-01-05

Figure 76 - Date Ranges Aggregation Rules - Analysis 9 Results

From the distinct Date Effective (2019-01-02) of discounts A and B to the Date Expired (2019-01-04) of discount A, both discounts share a common date range. From the Date Expired (+1) (2019-01-05) of discount A to the Date Effective (2019-01-05) of discount B, only the discount B is valid.

7.4.2.8. Analysis 10

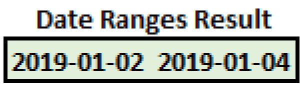
In this scenario, the Date Expired (+1) from discount A is analyzed in the event that there's another Date Expired (+1) occurring on the same date. This Date Expired (+1) will be considered once to correctly segregate the date ranges. Although in this context, since this outer Date Expired (+1) lies outside of the actual date ranges for discounts A and B, it will end up not being considered in the final solution, which only arranges the dates that actually have an advanced pricing agreement in place. Nonetheless, as part of the aggregation rules, it is considered.



Analysis nr 10a				🟪
Discount A	2019-01-01	**2019-01-02**	**2019-01-04**	2019-01-05
Discount B	2019-01-01	**2019-01-02**	**2019-01-04**	2019-01-05

Figure 77 - Date Ranges Aggregation Rules - Analysis 10a

The final solution with the date ranges that contain a discount is:

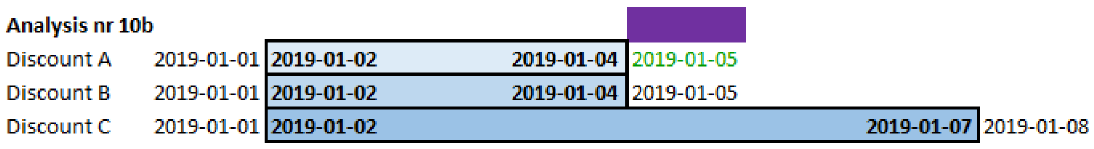


Date Ranges Result	
2019-01-02	2019-01-04

Figure 78 - Date Ranges Aggregation Rules - Analysis 10a Results

From the distinct Date Effective (2019-01-02) to the distinct Date Expired (2019-01-04) of discounts A and B, both discounts have a common date range.

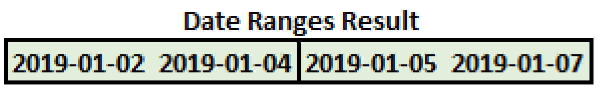
Considering the previous example, but adding a third discount C, it becomes clearer that this scenario must consider the Date Expired (+1).



Analysis nr 10b				🟪		
Discount A	2019-01-01	**2019-01-02**	**2019-01-04**	2019-01-05		
Discount B	2019-01-01	**2019-01-02**	**2019-01-04**	2019-01-05		
Discount C	2019-01-01	**2019-01-02**			**2019-01-07**	2019-01-08

Figure 79 - Date Ranges Aggregation Rules - Analysis 10b

The final solution with the date ranges that contain a discount is:



Date Ranges Result	
2019-01-02 2019-01-04	2019-01-05 2019-01-07

Figure 80 - Date Ranges Aggregation Rules - Analysis 10b Results

From the distinct Date Effective (2019-01-02) of discounts A, B and C to the distinct Date Expired (2019-01-04) of both discounts A and B, discounts A, B and C are valid. From the distinct Date Expired (+1) (2019-01-05) from discounts A and B to the Date Expired (2019-01-07) of discount C, only discount C is valid.

7.4.3. List Price Dates without Discount Dates

With all the scenarios combining discount dates having been analyzed, the list price will now be considered too. Assuming there's always a list price in place for a product, the first step is to analyze whether the list price Date Effective and Date Expired must be considered when there is no Foundation Discount Date nor Outer Discount Date overlapping that same date.

When to Insert List Price Dates Without Foundation Discounts Dates nor Outer Discount Dates			
Analysis nr	List Price	Discount	Action
11	Date Effective	no discount no outer	INSERT
12	Date Expired	no discount no outer	INSERT

Table 28 - When to Insert List Prices Without Foundation and Outer Discounts Dates

7.4.3.1. Analysis 11

In this scenario, the list price Date Effective is analyzed in the event that there's no Foundation Discount Date nor Outer Discount Date occurring on the same date. In these cases, the list price Date Effective will always be considered in order to correctly segregate the date ranges.

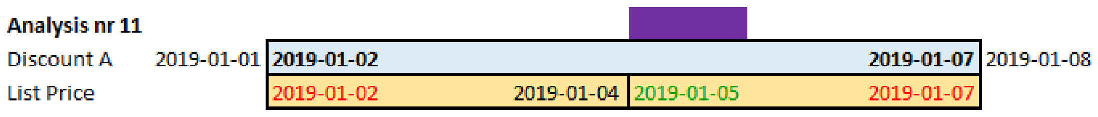
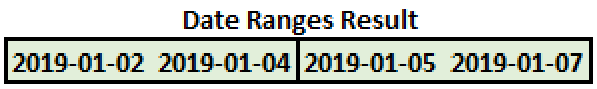


Figure 81 - Date Ranges Aggregation Rules - Analysis 11

The final solution with the date ranges that contain a discount is:



Date Ranges Result			
2019-01-02	2019-01-04	2019-01-05	2019-01-07

Figure 82 - Date Ranges Aggregation Rules - Analysis 11 Results

From the Date Effective (2019-01-02) of discount A to the Date Expired (2019-01-04) of the list price, discount A is valid with the list price of that same date range. From the Date Effective (2019-01-05) of the list price to the Date Expired (2019-01-07) of discount A, discount A is valid with the list price of that same date range.

7.4.3.2. Analysis 12

In this scenario, the list price Date Expired is analyzed in the event that there's no Foundation Discount Date nor Outer Discount Date occurring on the same date. In these cases, the list price Date Expired will always be considered in order to correctly segregate the date ranges.

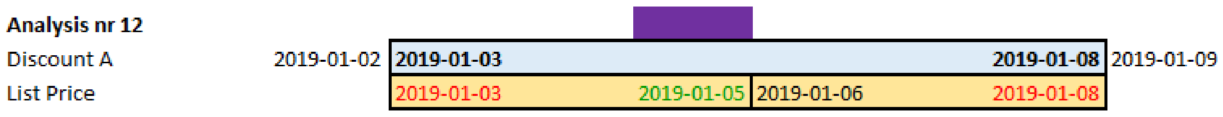
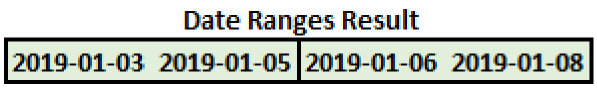


Figure 83 - Date Ranges Aggregation Rules - Analysis 12

The final solution with the date ranges that contain a discount is:



Date Ranges Result			
2019-01-03	2019-01-05	2019-01-06	2019-01-08

Figure 84 - Date Ranges Aggregation Rules - Analysis 12 Results

From the Date Effective (2019-01-03) of discount A to the Date Expired (2019-01-05) of the list price, discount A is valid with the list price of that same date range. From the Date Effective (2019-01-06) of the list price to the Date Expired (2019-01-08) of discount A, discount A is valid with the list price of that same date range.

7.4.4. List Price Dates with 1 Discount Date

Analysis 13 through 20 study the scenarios in which the List Price must be inserted when there is one discount date occurring on the same date.

When to Insert List Price Dates With 1 Discount Date			
Analysis nr	List Price	Discount A	Action
13	Date Effective	Date Effective (-1)	INSERT
14		Date Effective	DON'T INSERT
15		Date Expired	INSERT
16		Date Expired (+1)	DON'T INSERT
17	Date Expired	Date Effective (-1)	DON'T INSERT
18		Date Effective	INSERT
19		Date Expired	DON'T INSERT
20		Date Expired (+1)	INSERT

Table 29 - When to Insert List Price Dates With 1 Discount Date

7.4.4.1. Analysis 13

In this scenario, the list price Date Effective is analyzed in the event that there's a discount Date Effective (-1) occurring on the same date. In these cases, the list price Date Effective will always be considered in order to correctly segregate the date ranges.

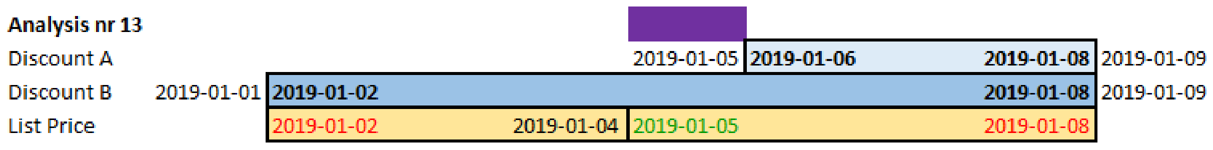
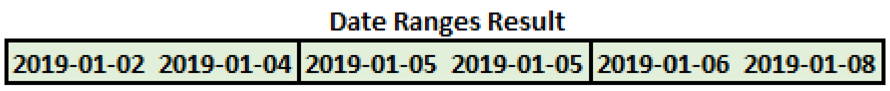


Figure 85 - Date Ranges Aggregation Rules - Analysis 13

The final solution with the date ranges that contain a discount is:



Date Ranges Result		
2019-01-02 2019-01-04	2019-01-05 2019-01-05	2019-01-06 2019-01-08

Figure 86 - Date Ranges Aggregation Rules - Analysis 13 Results

In case only discount A existed, the list price, although considered, wouldn't make a difference, since the valid date range from 2019-01-06 to 2019-01-08 wouldn't encompass the 2019-01-05 to 2019-01-05 period. Therefore, in order to better exemplify, a discount B was introduced to have a valid discount from the start to the end of the list price.

From the Date Effective (2019-01-02) of discount B to the Date Expired (2019-01-04) of the list price, discount B is valid with the list price of that same date range. From the Date Effective (2019-01-05) of the list price to the Date Effective (-1) (2019-01-05) of discount A, discount B is valid with the list price valid from 2019-01-05 to 2019-01-08. From the Date Effective (2019-01-06) of discount A to the distinct Date Expired (2019-01-08) of discounts A and B, both discounts are valid with the list price from 2019-01-05 to 2019-01-08.

7.4.4.2. Analysis 14

In this scenario, the list price Date Effective is analyzed in the event that there's a discount Date Effective occurring on the same date. In these cases, the list price Date Effective will never be considered in order to correctly segregate the date ranges.

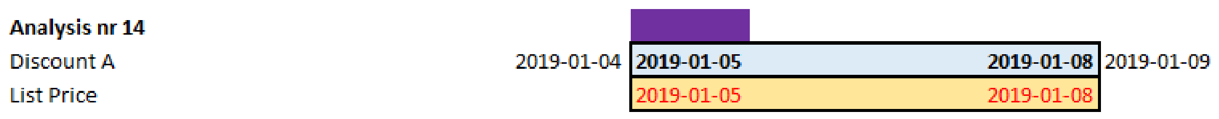
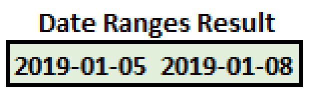


Figure 87 - Date Ranges Aggregation Rules - Analysis 14

The final solution with the date ranges that contain a discount is:



Date Ranges Result
2019-01-05
2019-01-08

Figure 88 - Date Ranges Aggregation Rules - Analysis 14 Results

From the Date Effective (2019-01-05) to the Date Expired (2019-01-08) of discount A, the discount is valid with the list price of that same date range.

7.4.4.3. Analysis 15

In this scenario, the list price Date Effective is analyzed in the event that there's a discount Date Expired occurring on the same date. In these cases, the list price Date Effective will always be considered in order to correctly segregate the date ranges.

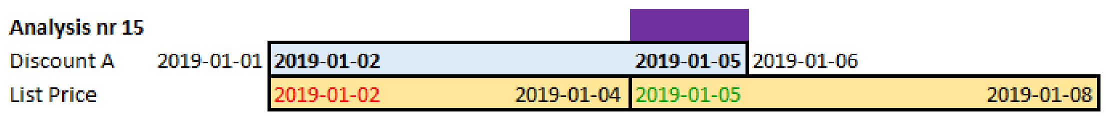
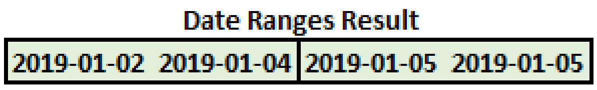


Figure 89 - Date Ranges Aggregation Rules - Analysis 15

The final solution with the date ranges that contain a discount is:



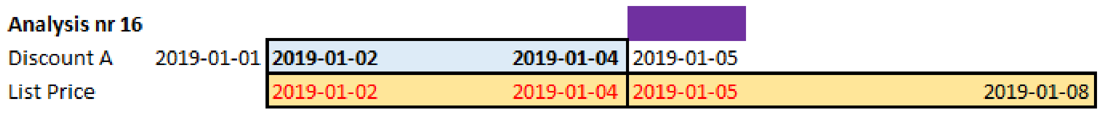
Date Ranges Result			
2019-01-02	2019-01-04	2019-01-05	2019-01-05

Figure 90 - Date Ranges Aggregation Rules - Analysis 15 Results

From the Date Effective (2019-01-02) of discount A to the Date Expired (2019-01-04) of the list price, discount A is valid with the list price from 2019-01-02 to 2019-01-04. From the Date Expired (2019-01-05) of discount A to the Date Effective (2019-01-05) of the list price, discount A is valid with the list price valid from 2019-01-05 to 2019-01-08.

7.4.4.4. Analysis 16

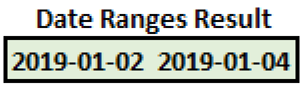
In this scenario, the list price Date Effective is analyzed in the event that there's a discount Date Expired (+1) occurring on the same date. In these cases, the list price Date Effective will never be considered in order to correctly segregate the date ranges.



Analysis nr 16				🟪	
Discount A	2019-01-01	2019-01-02	2019-01-04	2019-01-05	
List Price		2019-01-02	2019-01-04	2019-01-05	2019-01-08

Figure 91 - Date Ranges Aggregation Rules - Analysis 16

The final solution with the date ranges that contain a discount is:



Date Ranges Result	
2019-01-02	2019-01-04

Figure 92 - Date Ranges Aggregation Rules - Analysis 16 Results

From the Date Effective (2019-01-02) to the Date Expired (2019-01-04) of discount A, the discount is valid with the list price of that same date range.

7.4.4.5. Analysis 17

In this scenario, the list price Date Expired is analyzed in the event that there's a discount Date Effective (-1) occurring on the same date. In these cases, the list price Date Expired will never be considered in order to correctly segregate the date ranges.

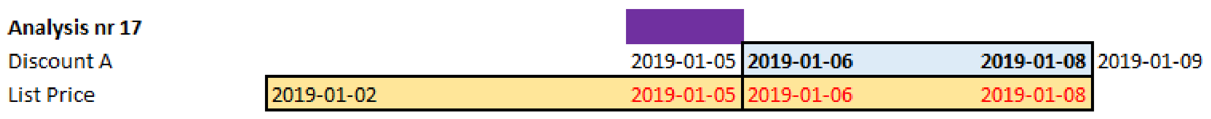


Figure 93 - Date Ranges Aggregation Rules - Analysis 17

The final solution with the date ranges that contain a discount is:

Date Ranges Result	
2019-01-06	2019-01-08

Figure 94 - Date Ranges Aggregation Rules - Analysis 17 Results

From the Date Effective (2019-01-06) to the Date Expired (2019-01-08) of discount A, the discount is valid with the list price of that same date range.

7.4.4.6. Analysis 18

In this scenario, the list price Date Expired is analyzed in the event that there’s a discount Date Effective occurring on the same date. In these cases, the list price Date Expired will always be considered in order to correctly segregate the date ranges.

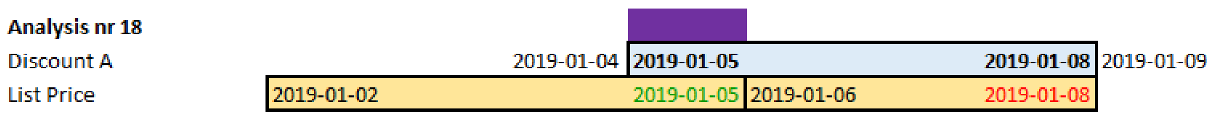


Figure 95 - Date Ranges Aggregation Rules - Analysis 18

The final solution with the date ranges that contain a discount is:

Date Ranges Result			
2019-01-05	2019-01-05	2019-01-06	2019-01-08

Figure 96 - Date Ranges Aggregation Rules - Analysis 18 Results

From the Date Effective (2019-01-05) of discount A to the Date Expired (2019-01-05) of the list price, discount A is valid with the list price valid from 2019-01-02 to 2019-01-05. From the Date Effective (2019-01-06) of the list price to the Date Expired (2019-01-08) of discount A, discount A is valid with the list price from 2019-01-06 to 2019-01-08.

7.4.4.7. Analysis 19

In this scenario, the list price Date Expired is analyzed in the event that there's a discount Date Expired occurring on the same date. In these cases, the list price Date Expired will never be considered in order to correctly segregate the date ranges.

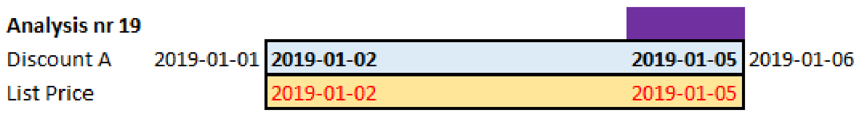
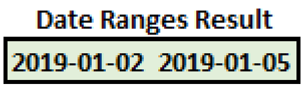


Figure 97 - Date Ranges Aggregation Rules - Analysis 19

The final solution with the date ranges that contain a discount is:



Date Ranges Result	
2019-01-02	2019-01-05

Figure 98 - Date Ranges Aggregation Rules - Analysis 19 Results

From the Date Effective (2019-01-02) to the Date Expired (2019-01-05) of discount A, the discount is valid with the list price of that same date range.

7.4.4.8. Analysis 20

In this scenario, the list price Date Expired is analyzed in the event that there's a discount Date Expired (+1) occurring on the same date. In these cases, the list price Date Expired will always be considered in order to correctly segregate the date ranges.

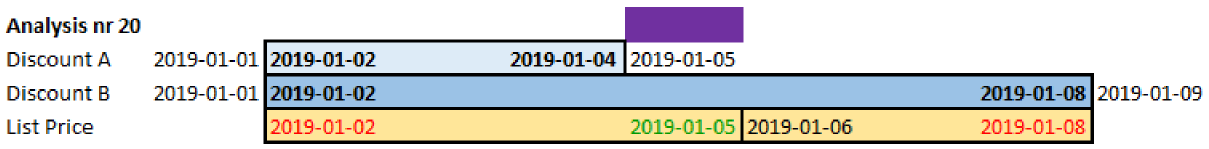
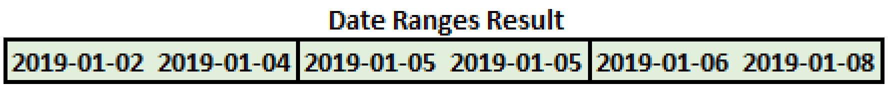


Figure 99 - Date Ranges Aggregation Rules - Analysis 20

The final solution with the date ranges that contain a discount is:



Date Ranges Result					
2019-01-02	2019-01-04	2019-01-05	2019-01-05	2019-01-06	2019-01-08

Figure 100 - Date Ranges Aggregation Rules - Analysis 20 Results

In case only discount A existed, the list price, although considered, wouldn't make a difference, since the valid date range from 2019-01-02 to 2019-01-04 wouldn't encompass the 2019-01-05 to 2019-01-05 period. Therefore, in order to better exemplify, a discount B was introduced to have a valid discount from the start to the end of the list price.

From the distinct Date Effective (2019-01-02) of discounts A and B to the Date Expired (2019-01-04) of discount A, both discounts A and B are valid with the list price of 2019-01-02 to 2019-01-05. From the Date Expired (+1) (2019-01-05) of Discount A to the Date Expired (2019-01-05) of the list price, discount B is valid with the list price valid from 2019-01-02 to 2019-01-05. From the Date Effective (2019-01-06) of the list price to the Date Expired (2019-01-08) of discount B, discount B is valid with the list price of that same date range.

7.4.5. List Price Dates with 2 Discount Dates

The analysis 21 to 32 study the scenarios in which a List Price date must be inserted when there are 2 Discount Dates occurring on the same date.

When to Insert List Price Dates With 2 Discount Dates				
Analysis nr	**List Price**	**Discount A**	**Discount B**	**Action**
21	Date Effective	Date Effective (-1)	Date Effective	DON'T INSERT
22		Date Effective (-1)	Date Expired	INSERT
23		Date Effective (-1)	Date Expired (+1)	DON'T INSERT
24		Date Effective	Date Expired	DON'T INSERT
25		Date Effective	Date Expired (+1)	DON'T INSERT
26		Date Expired	Date Expired (+1)	DON'T INSERT
27	Date Expired	Date Effective (-1)	Date Effective	DON'T INSERT
28		Date Effective (-1)	Date Expired	DON'T INSERT
29		Date Effective (-1)	Date Expired (+1)	DON'T INSERT
30		Date Effective	Date Expired	DON'T INSERT
31		Date Effective	Date Expired (+1)	INSERT
32		Date Expired	Date Expired (+1)	DON'T INSERT

Table 30 - When to Insert List Price Dates With 2 Discount Dates

7.4.5.1. Analysis 21

In this scenario, the list price Date Effective is analyzed in the event that there's a discount Date Effective (-1) and Date Effective occurring on the same date. In these cases, the list price Date Effective will never be considered in order to correctly segregate the date ranges.

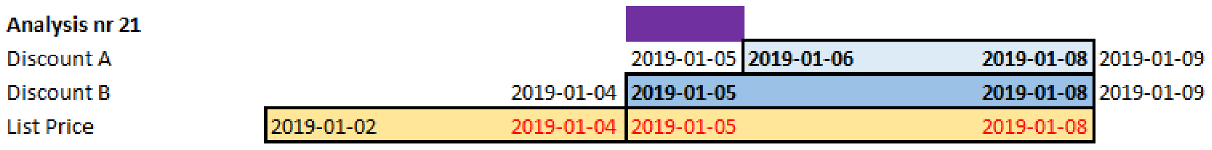


Figure 101 - Date Ranges Aggregation Rules - Analysis 21

The final solution with the date ranges that contain a discount is:

Date Ranges Result

2019-01-05	2019-01-05	2019-01-06	2019-01-08

Figure 102 - Date Ranges Aggregation Rules - Analysis 21 Results

From the Date Effective (-1) (2019-01-05) of discount A to the Date Effective (2019-01-05) of discount B, discount B is valid with the list price valid from 2019-01-05 to 2019-01-08. From the Date Effective (2019-01-06) of discount A to the distinct Date Expired (2019-01-08) of discounts A and B, both discounts are valid with the list price from 2019-01-05 to 2019-01-08.

7.4.5.2. Analysis 22

In this scenario, the list price Date Effective is analyzed in the event that there's a discount Date Effective (-1) and Date Expired occurring on the same date. In these cases, the list price Date Effective will always be considered in order to correctly segregate the date ranges.

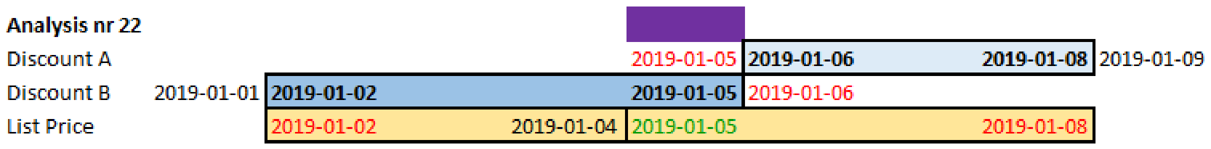


Figure 103 - Date Ranges Aggregation Rules - Analysis 22

The final solution with the date ranges that contain a discount is:

Date Ranges Result

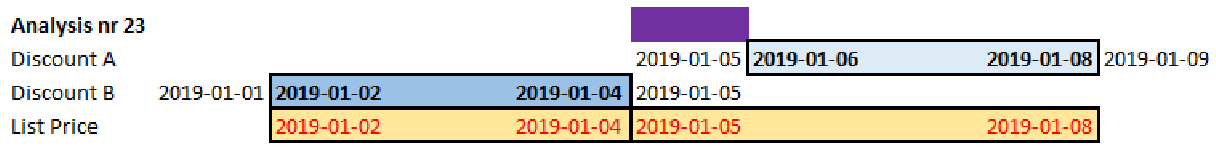
2019-01-02	2019-01-04	2019-01-05	2019-01-05	2019-01-06	2019-01-08

Figure 104 - Date Ranges Aggregation Rules - Analysis 22 Results

From the Date Effective (2019-01-02) of discount B to the Date Expired (2019-01-04) of the list price, discount B is valid with the list price of 2019-01-02 to 2019-01-04. From the Date Expired (2019-01-05) of discount B to the Date Effective (2019-01-05) of the list price, discount B is valid with the list price valid from 2019-01-05 to 2019-01-08. From the Date Effective (2019-01-06) to the Date Expired (2019-01-08) of discount A, discount A is valid with the list price from 2019-01-05 to 2019-01-08.

7.4.5.3. Analysis 23

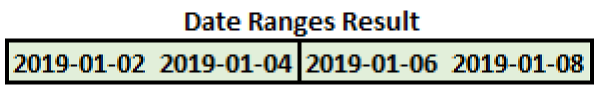
In this scenario, the list price Date Effective is analyzed in the event that there's a discount Date Effective (-1) and Date Expired (+1) occurring on the same date. In these cases, the list price Date Effective will never be considered in order to correctly segregate the date ranges.



Analysis nr 23							
Discount A				2019-01-05	**2019-01-06**	**2019-01-08**	2019-01-09
Discount B	2019-01-01	**2019-01-02**	**2019-01-04**	2019-01-05			
List Price		2019-01-02	2019-01-04	2019-01-05		2019-01-08	

Figure 105 - Date Ranges Aggregation Rules - Analysis 23

The final solution with the date ranges that contain a discount is:



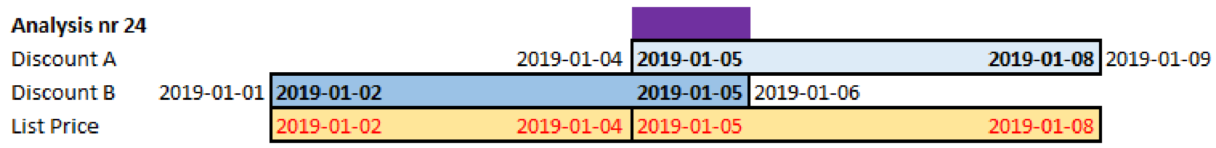
Date Ranges Result			
2019-01-02	2019-01-04	2019-01-06	2019-01-08

Figure 106 - Date Ranges Aggregation Rules - Analysis 23 Results

From the Date Effective (2019-01-02) to the Date Expired (2019-01-04) of discount B, discount B is valid with the list price of that same date range. From the Date Effective (2019-01-06) to the Date Expired (2019-01-08) of discount A, discount A is valid with the list price from 2019-01-05 to 2019-01-08.

7.4.5.4. Analysis 24

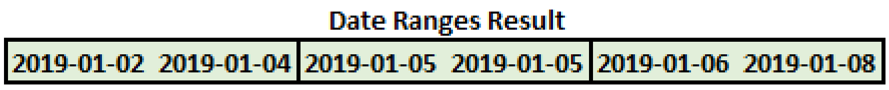
In this scenario, the list price Date Effective is analyzed in the event that there's a discount Date Effective and Date Expired occurring on the same date. In these cases, the list price Date Effective will never be considered in order to correctly segregate the date ranges.



Analysis nr 24							
Discount A			2019-01-04	**2019-01-05**		**2019-01-08**	2019-01-09
Discount B	2019-01-01	**2019-01-02**		**2019-01-05**	2019-01-06		
List Price		2019-01-02	2019-01-04	2019-01-05		2019-01-08	

Figure 107 - Date Ranges Aggregation Rules - Analysis 24

The final solution with the date ranges that contain a discount is:



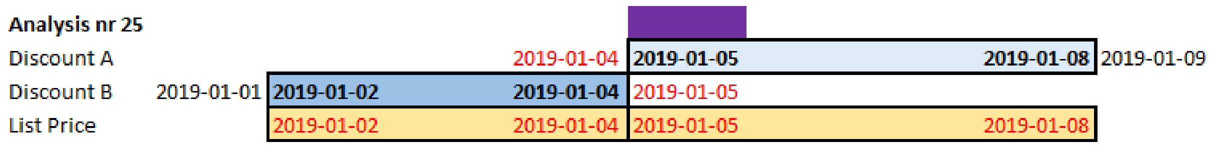
Date Ranges Result					
2019-01-02	2019-01-04	2019-01-05	2019-01-05	2019-01-06	2019-01-08

Figure 108 - Date Ranges Aggregation Rules - Analysis 24 Results

From the Date Effective (2019-01-02) of discount B to the Date effective (-1) (2019-01-04) of discount A, discount B is valid with the list price of 2019-01-02 to 2019-01-04. From the Date Effective (2019-01-05) of discount A to the Date Expired (2019-01-05) of discount B, both discounts are valid with the list price valid from 2019-01-05 to 2019-01-08. From the Date Expired (+1) (2019-01-06) of discount B to the Date Expired (2019-01-08) of discount A, discount A is valid with the list price from 2019-01-05 to 2019-01-08.

7.4.5.5. Analysis 25

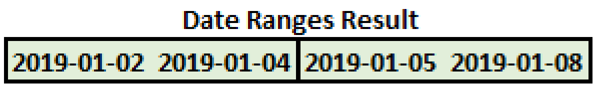
In this scenario, the list price Date Effective is analyzed in the event that there's a discount Date Effective and Date Expired (+1) occurring on the same date. In these cases, the list price Date Effective will never be considered in order to correctly segregate the date ranges.



Analysis nr 25						
Discount A			2019-01-04	2019-01-05	2019-01-08	2019-01-09
Discount B	2019-01-01	2019-01-02	2019-01-04	2019-01-05		
List Price		2019-01-02	2019-01-04	2019-01-05	2019-01-08	

Figure 109 - Date Ranges Aggregation Rules - Analysis 25

The final solution with the date ranges that contain a discount is:



Date Ranges Result			
2019-01-02	2019-01-04	2019-01-05	2019-01-08

Figure 110 - Date Ranges Aggregation Rules - Analysis 25 Results

From the Date Effective (2019-01-02) to the Date Expired (2019-01-04) of discount B, discount B is valid with the list price of that same date range. From the Date Effective (2019-01-05) to the Date Expired (2019-01-08) of discount A, discount A is valid with the list price from 2019-01-05 to 2019-01-08.

7.4.5.6. Analysis 26

In this scenario, the list price Date Effective is analyzed in the event that there's a discount Date Expired and Date Expired (+1) occurring on the same date. In these cases, the list price Date Effective will never be considered in order to correctly segregate the date ranges.

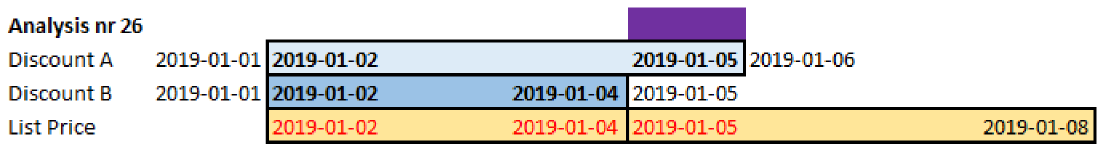
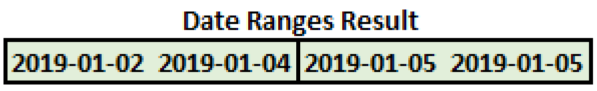


Figure 111 - Date Ranges Aggregation Rules - Analysis 26

The final solution with the date ranges that contain a discount is:



Date Ranges Result
2019-01-02 2019-01-04
2019-01-05 2019-01-05

Figure 112 - Date Ranges Aggregation Rules - Analysis 26 Results

From the distinct Date Effective (2019-01-02) of discounts A and B to the Date Expired (2019-01-04) of discount B, both discounts are valid with the list price of 2019-01-02 to 2019-01-04. From the Date Expired (2019-01-05) of discount A to the Date Expired (+1) (2019-01-05) of discount B, discount A is valid with the list price valid from 2019-01-05 to 2019-01-08.

7.4.5.7. Analysis 27

In this scenario, the list price Date Expired is analyzed in the event that there's a discount Date Effective (-1) and Date Effective occurring on the same date. In these cases, the list price Date Expired will never be considered in order to correctly segregate the date ranges.

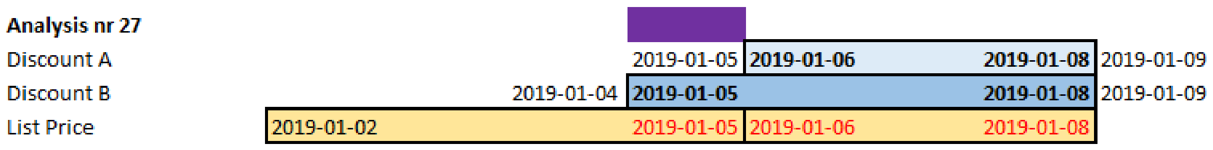
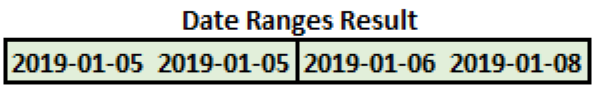


Figure 113 - Date Ranges Aggregation Rules - Analysis 27

The final solution with the date ranges that contain a discount is:



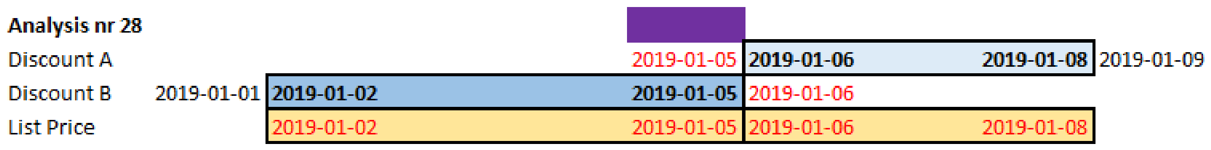
Date Ranges Result			
2019-01-05	2019-01-05	2019-01-06	2019-01-08

Figure 114 - Date Ranges Aggregation Rules - Analysis 27 Results

From the Date Effective (-1) (2019-01-05) of discount A to the Date Effective (2019-01-05) of discount B, discount B is valid with the list price valid from 2019-01-02 to 2019-01-05. From the Date Effective (2019-01-06) of discount A to the distinct Date Expired (2019-01-08) of discounts A and B, both discounts are valid with the list price from 2019-01-06 to 2019-01-08

7.4.5.8. Analysis 28

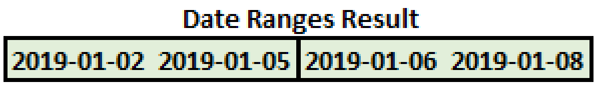
In this scenario, the list price Date Expired is analyzed in the event that there's a discount Date Effective (-1) and Date Expired occurring on the same date. In these cases, the list price Date Expired will never be considered in order to correctly segregate the date ranges.



Analysis nr 28						
Discount A			2019-01-05	2019-01-06	2019-01-08	2019-01-09
Discount B	2019-01-01	2019-01-02	2019-01-05	2019-01-06		
List Price		2019-01-02	2019-01-05	2019-01-06	2019-01-08	

Figure 115 - Date Ranges Aggregation Rules - Analysis 28

The final solution with the date ranges that contain a discount is:



Date Ranges Result			
2019-01-02	2019-01-05	2019-01-06	2019-01-08

Figure 116 - Date Ranges Aggregation Rules - Analysis 28 Results

From the Date Effective (2019-01-02) to the Date Expired (2019-01-05) of discount B, discount B is valid with the list price of that same date range. From the Date Effective (2019-01-06) to the Date Expired (2019-01-08) of discount A, discount A is valid with the list price from that same date range.

7.4.5.9. Analysis 29

In this scenario, the list price Date Expired is analyzed in the event that there's a discount Date Effective (-1) and Date Expired (+1) occurring on the same date. In these cases, the list price Date Expired will never be considered in order to correctly segregate the date ranges.

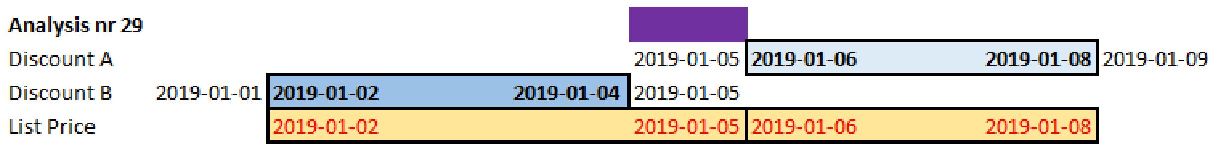


Figure 117 - Date Ranges Aggregation Rules - Analysis 29

The final solution with the date ranges that contain a discount is:

Date Ranges Result

2019-01-02 2019-01-04	2019-01-06 2019-01-08

Figure 118 - Date Ranges Aggregation Rules - Analysis 29 Results

From the Date Effective (2019-01-02) to the Date Expired (2019-01-04) of discount B, discount B is valid with the list price of 2019-01-02 to 2019-01-05. From the Date Effective (2019-01-06) to the Date Expired (2019-01-08) of discount A, discount A is valid with the list price from that same date range.

7.4.5.10. Analysis 30

In this scenario, the list price Date Expired is analyzed in the event that there's a discount Date Effective and Date Expired occurring on the same date. In these cases, the list price Date Expired will never be considered in order to correctly segregate the date ranges.

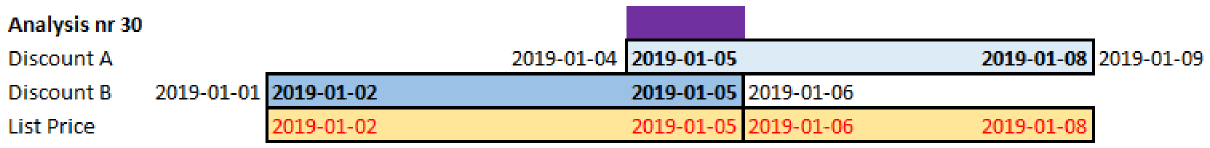


Figure 119 - Date Ranges Aggregation Rules - Analysis 30

The final solution with the date ranges that contain a discount is:

Date Ranges Result

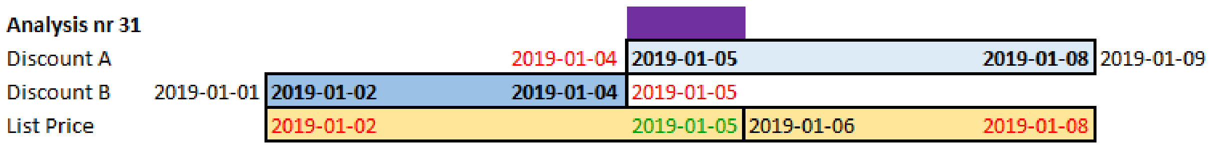
2019-01-02 2019-01-04	2019-01-05 2019-01-05	2019-01-06 2019-01-08

Figure 120 - Date Ranges Aggregation Rules - Analysis 30 Results

From the Date Effective (2019-01-02) of discount B to the Date Effective (-1) (2019-01-04) of discount A, discount B is valid with the list price of 2019-01-02 to 2019-01-05. From the Date Effective (2019-01-05) of discount A to the Date Expired (2019-01-05) of discount B, both discounts are valid with the list price valid from 2019-01-02 to 2019-01-05. From the Date Expired (+1) (2019-01-06) of discount B to the Date Expired (2019-01-08) of discount A, discount A is valid with the list price from 2019-01-06 to 2019-01-08.

7.4.5.11. Analysis 31

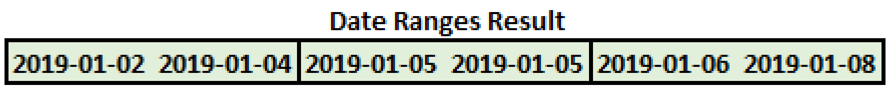
In this scenario, the list price Date Expired is analyzed in the event that there's a discount Date Effective and Date Expired (+1) occurring on the same date. In these cases, the list price Date Expired will always be considered in order to correctly segregate the date ranges.



Analysis nr 31							
Discount A			2019-01-04	**2019-01-05**		**2019-01-08**	2019-01-09
Discount B	2019-01-01	**2019-01-02**	**2019-01-04**	2019-01-05			
List Price		2019-01-02		2019-01-05	2019-01-06	2019-01-08	

Figure 121 - Date Ranges Aggregation Rules - Analysis 31

The final solution with the date ranges that contain a discount is:



Date Ranges Result

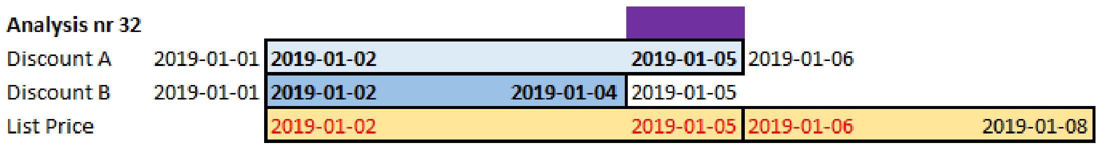
2019-01-02	2019-01-04	2019-01-05	2019-01-05	2019-01-06	2019-01-08

Figure 122 - Date Ranges Aggregation Rules - Analysis 31 Results

From the Date Effective (2019-01-02) to the Date Expired (2019-01-04) of discount B, discount B is valid with the list price of 2019-01-02 to 2019-01-05. From the Date Effective (2019-01-05) of discount A to the Date Expired (2019-01-05) of the list price, discount A is valid with the list price from 2019-01-02 to 2019-01-05. From the Date Effective (2019-01-06) of the list price to the Date Expired (2019-01-08) of discount A, discount A is valid with the list price from 2019-01-06 to 2019-01-08.

7.4.5.12. Analysis 32

In this scenario, the list price Date Expired is analyzed in the event that there's a discount Date Expired and Date Expired (+1) occurring on the same date. In these cases, the list price Date Expired will never be considered in order to correctly segregate the date ranges.



Analysis nr 32						
Discount A	2019-01-01	**2019-01-02**		**2019-01-05**	2019-01-06	
Discount B	2019-01-01	**2019-01-02**	**2019-01-04**	2019-01-05		
List Price		2019-01-02		2019-01-05	2019-01-06	2019-01-08

Figure 123 - Date Ranges Aggregation Rules - Analysis 32

The final solution with the date ranges that contain a discount is:

Date Ranges Result	
2019-01-02 2019-01-04	2019-01-05 2019-01-05

Figure 124 - Date Ranges Aggregation Rules – Analysis 32 Results

From the distinct Date Effective (2019-01-02) of discounts A and B to the Date Expired (2019-01-04) of discount B, both discounts are valid with the list price of 2019-01-02 to 2019-01-05. From the Date Expired (2019-01-05) of discount A to the Date Expired (+1) (2019-01-05) of discount B, discount A is valid with the list price valid from 2019-01-02 to 2019-01-05.

7.4.6. List Price Dates with 3 Discount Dates

Analyzing all the possible scenarios using 3 discount dates and a list price date at the same time, the list price date will never be considered.

When to Insert List Price Dates With 3 Discount Dates					
Analysis nr	List Price	Discount A	Discount B	Discount C	Action
33	Date Effective	Date Effective (-1)	Date Effective	Date Expired	DON'T INSERT
34		Date Effective (-1)	Date Effective	Date Expired (+1)	DON'T INSERT
35		Date Effective (-1)	Date Expired	Date Expired (+1)	DON'T INSERT
36		Date Effective	Date Expired	Date Expired (+1)	DON'T INSERT
37	Date Expired	Date Effective (-1)	Date Effective	Date Expired	DON'T INSERT
38		Date Effective (-1)	Date Effective	Date Expired (+1)	DON'T INSERT
39		Date Effective (-1)	Date Expired	Date Expired (+1)	DON'T INSERT
40		Date Effective	Date Expired	Date Expired (+1)	DON'T INSERT

Table 31 - When to Insert List Price Dates With 3 Discount Dates

7.4.6.1. Analysis 33

In this scenario, the list price Date Effective is analyzed in the event that there's a discount Date Effective (-1), Date Effective and Date Expired occurring on the same date. In these cases, the list price Date Effective will never be considered in order to correctly segregate the date ranges.

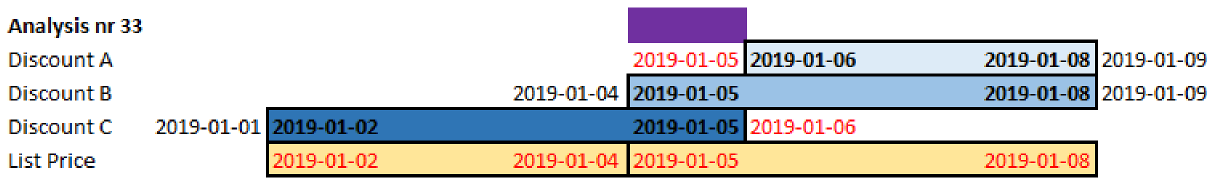


Figure 125 - Date Ranges Aggregation Rules - Analysis 33

The final solution with the date ranges that contain a discount is:

Date Ranges Result

2019-01-02	2019-01-04	2019-01-05	2019-01-05	2019-01-06	2019-01-08

Figure 126 - Date Ranges Aggregation Rules - Analysis 33 Results

From the Date Effective (2019-01-02) of discount C to the Date Effective (-1) (2019-01-04) of discount B, discount C is valid with the list price from 2019-01-02 to 2019-01-04. From the Date Effective (2019-01-05) of discount B to the Date Expired (2019-01-05) of discount C, discounts B and C are valid with the list price from 2019-01-05 to 2019-01-08. From the Date Effective (2019-01-06) of discount A to the distinct Date Expired (2019-01-08) of discounts A and B, discounts A and B are valid with the list price from 2019-01-05 to 2019-01-08.

7.4.6.2. Analysis 34

In this scenario, the list price Date Effective is analyzed in the event that there's a discount Date Effective (-1), Date Effective and Date Expired (+1) occurring on the same date. In these cases, the list price Date Effective will never be considered in order to correctly segregate the date ranges.

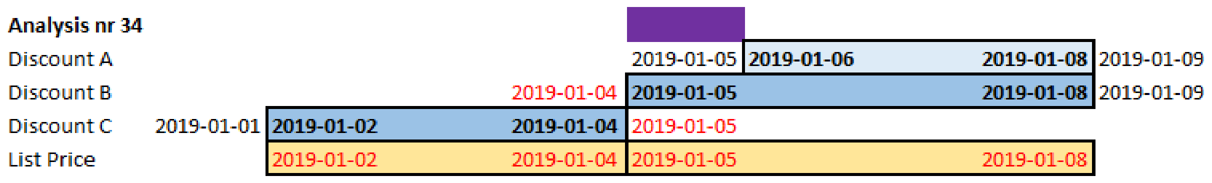


Figure 127 - Date Ranges Aggregation Rules Analysis 34

The final solution with the date ranges that contain a discount is:

Date Ranges Result

2019-01-02	2019-01-04	2019-01-05	2019-01-05	2019-01-06	2019-01-08

Figure 128 - Date Ranges Aggregation Rules - Analysis 34 Results

From the Date Effective (2019-01-02) to the Date Expired (2019-01-04) of discount C, discount C is valid with the list price from 2019-01-02 to 2019-01-04. From the Date Effective (-1) (2019-01-05) of discount A to the Date Effective (2019-01-05) of discount B, discount B is valid with the list price from 2019-01-05 to 2019-01-08. From the Date Effective (2019-01-06) of discount A to the distinct Date Expired (2019-01-08) of discounts A and B, discounts A and B are valid with the list price from 2019-01-05 to 2019-01-08.

7.4.6.3. Analysis 35

In this scenario, the list price Date Effective is analyzed in the event that there's a discount Date Effective (-1), Date Expired and Date Expired (+1) occurring on the same date. In these cases, the list price Date Effective will never be considered in order to correctly segregate the date ranges.

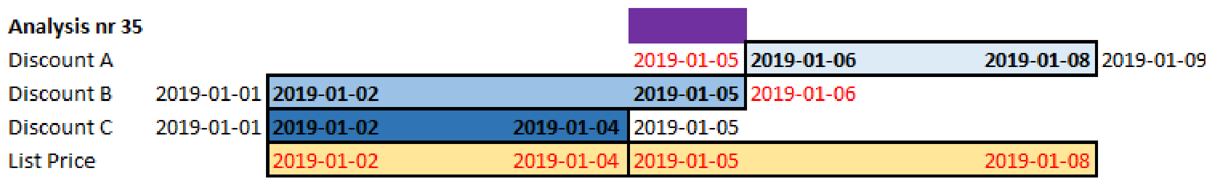
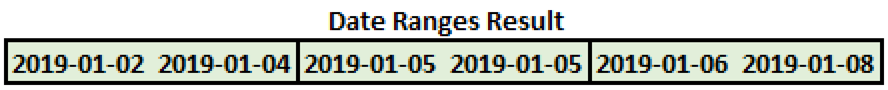


Figure 129 - Date Ranges Aggregation Rules Analysis 35

The final solution with the date ranges that contain a discount is:



Date Ranges Result

2019-01-02	2019-01-04	2019-01-05	2019-01-05	2019-01-06	2019-01-08

Figure 130 - Date Ranges Aggregation Rules - Analysis 35 Results

From the distinct Date Effective (2019-01-02) of discounts B and C to the Date Expired (2019-01-04) of discount C, discounts B and C are valid with the list price from 2019-01-02 to 2019-01-04. From the Date Expired (2019-01-05) of discount B to the Date Expired (+1) (2019-01-05) of discount C, discount B is valid with the list price from 2019-01-05 to 2019-01-08. From the Date Effective (2019-01-06) to the Date Expired (2019-01-08) of discount A, discount A is valid with the list price from 2019-01-05 to 2019-01-08.

7.4.6.4. Analysis 36

In this scenario, the list price Date Effective is analyzed in the event that there's a discount Date Effective, Date Expired and Date Expired (+1) occurring on the same date. In these cases, the list price Date Effective will never be considered in order to correctly segregate the date ranges.

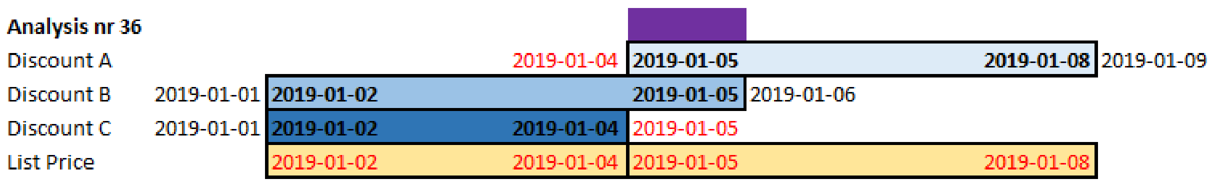
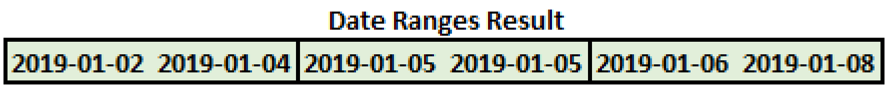

Analysis nr 36
Discount A 2019-01-04 2019-01-05 2019-01-08 2019-01-09
Discount B 2019-01-01 2019-01-02 2019-01-05 2019-01-06
Discount C 2019-01-01 2019-01-02 2019-01-04 2019-01-05
List Price 2019-01-02 2019-01-04 2019-01-05 2019-01-08

Figure 131 - Date Ranges Aggregation Rules Analysis 36

The final solution with the date ranges that contain a discount is:



Date Ranges Result

2019-01-02	2019-01-04	2019-01-05	2019-01-05	2019-01-06	2019-01-08

Figure 132 - Date Ranges Aggregation Rules - Analysis 36 Results

From the distinct Date Effective (2019-01-02) of discounts B and C to the Date Expired (2019-01-04) of discount C, discounts B and C are valid with the list price from 2019-01-02 to 2019-01-04. From the Date Effective (2019-01-05) of discount A to the Date Expired (2019-01-05) of discount B, discounts A and B are valid with the list price from 2019-01-05 to 2019-01-08. From the Date Effective (+1) (2019-01-06) of discount B to the Date Expired (2019-01-08) of discount A, discount A is valid with the list price from 2019-01-05 to 2019-01-08.

7.4.6.5. Analysis 37

In this scenario, the list price Date Expired is analyzed in the event that there's a discount Date Effective (-1), Date Effective and Date Expired occurring on the same date. In these cases, the list price Date Expired will never be considered in order to correctly segregate the date ranges.

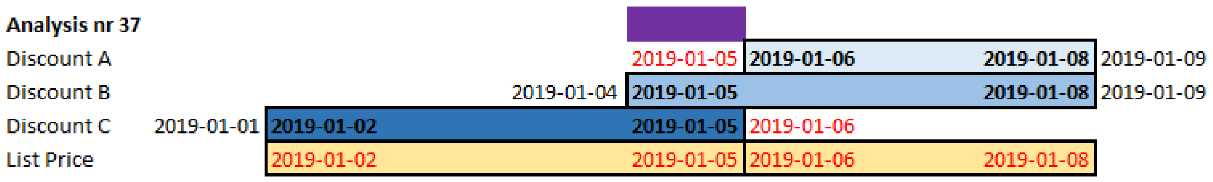
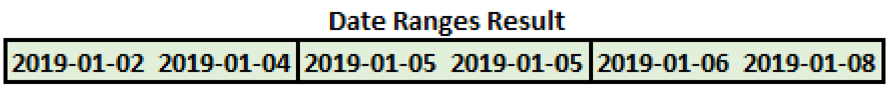

Analysis nr 37
Discount A 2019-01-05 2019-01-06 2019-01-08 2019-01-09
Discount B 2019-01-04 2019-01-05 2019-01-08 2019-01-09
Discount C 2019-01-01 2019-01-02 2019-01-05 2019-01-06
List Price 2019-01-02 2019-01-05 2019-01-06 2019-01-08

Figure 133 - Date Ranges Aggregation Rules Analysis 37

The final solution with the date ranges that contain a discount is:



Date Ranges Result

2019-01-02	2019-01-04	2019-01-05	2019-01-05	2019-01-06	2019-01-08

Figure 134 - Date Ranges Aggregation Rules - Analysis 37 Results

From the Date Effective (2019-01-02) of discount C to the Date Effective (-1) (2019-01-04) of discount B, discount C is valid with the list price from 2019-01-02 to 2019-01-05. From the Date Effective (2019-01-05) of discount B to the Date Expired (2019-01-05) of discount C, discounts B and C are valid with the list price from 2019-01-02 to 2019-01-05. From the Date Effective (2019-01-06) of discount A to the distinct Date Expired (2019-01-08) of discounts A and B, discounts A and B are valid with the list price from 2019-01-06 to 2019-01-08.

7.4.6.6. Analysis 38

In this scenario, the list price Date Expired is analyzed in the event that there's a discount Date Effective (-1), Date Effective and Date Expired (+1) occurring on the same date. In these cases, the list price Date Expired will never be considered in order to correctly segregate the date ranges.

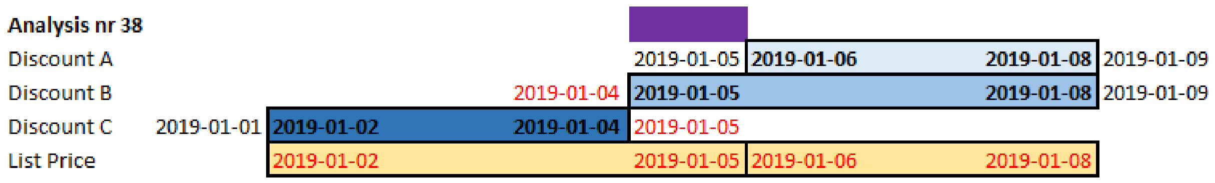
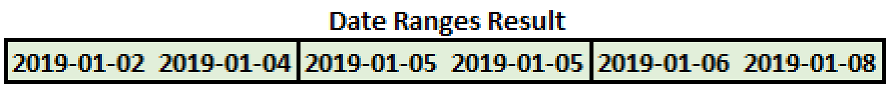


Figure 135 - Date Ranges Aggregation Rules Analysis 38

The final solution with the date ranges that contain a discount is:



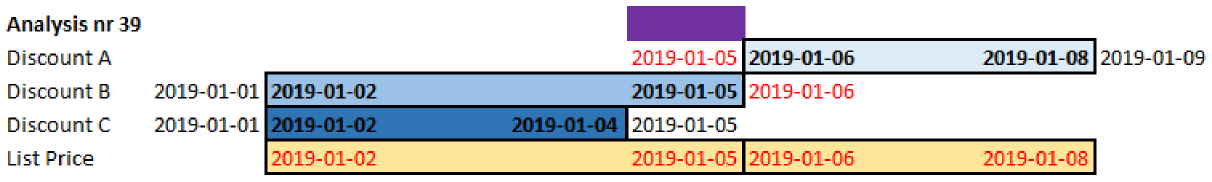
Date Ranges Result
2019-01-02 2019-01-04
2019-01-05 2019-01-05
2019-01-06 2019-01-08

Figure 136 - Date Ranges Aggregation Rules - Analysis 38 Results

From the Date Effective (2019-01-02) to the Date Expired (2019-01-04) of discount C, discount C is valid with the list price from 2019-01-02 to 2019-01-05. From the Date Effective (-1) (2019-01-05) of discount A to the Date Effective (2019-01-05) of discount B, discount B is valid with the list price from 2019-01-02 to 2019-01-05. From the Date Effective (2019-01-06) of discount A to the distinct Date Expired (2019-01-08) of discounts A and B, discounts A and B are valid with the list price from 2019-01-06 to 2019-01-08.

7.4.6.7. Analysis 39

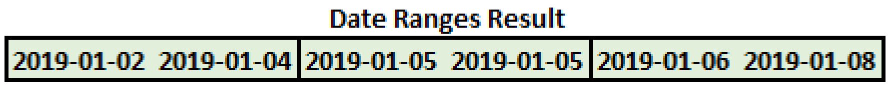
In this scenario, the list price Date Expired is analyzed in the event that there's a discount Date Effective (-1), Date Expired and Date Expired (+1) occurring on the same date. In these cases, the list price Date Expired will never be considered in order to correctly segregate the date ranges.



Analysis nr 39							
Discount A				2019-01-05	2019-01-06	2019-01-08	2019-01-09
Discount B	2019-01-01	2019-01-02		2019-01-05	2019-01-06		
Discount C	2019-01-01	2019-01-02	2019-01-04	2019-01-05			
List Price		2019-01-02		2019-01-05	2019-01-06	2019-01-08	

Figure 137 - Date Ranges Aggregation Rules Analysis 39

The final solution with the date ranges that contain a discount is:



Date Ranges Result

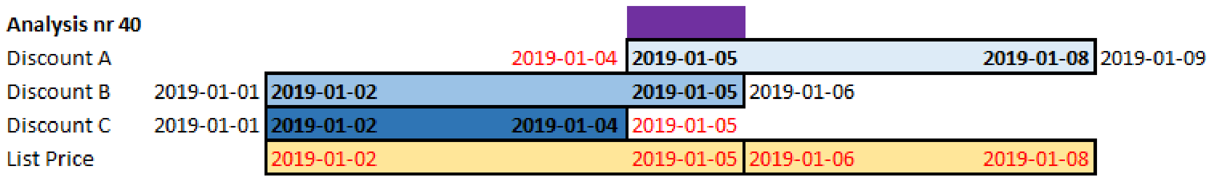
2019-01-02	2019-01-04	2019-01-05	2019-01-05	2019-01-06	2019-01-08

Figure 138 - Date Ranges Aggregation Rules - Analysis 39 Results

From the distinct Date Effective (2019-01-02) of discounts B and C to the Date Expired (2019-01-04) of discount C, discounts B and C are valid with the list price from 2019-01-02 to 2019-01-05. From the Date Expired (2019-01-05) of discount B to the Date Expired (+1) (2019-01-05) of discount C, discount B is valid with the list price from 2019-01-02 to 2019-01-05. From the Date Effective (2019-01-06) to the Date Expired (2019-01-08) of discount A, discount A is valid with the list price from 2019-01-06 to 2019-01-08.

7.4.6.8. Analysis 40

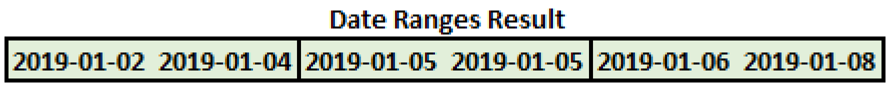
In this scenario, the list price Date Expired is analyzed in the event that there's a discount Date Effective, Date Expired and Date Expired (+1) occurring on the same date. In these cases, the list price Date Expired will never be considered in order to correctly segregate the date ranges.



Analysis nr 40							
Discount A			2019-01-04	2019-01-05		2019-01-08	2019-01-09
Discount B	2019-01-01	2019-01-02		2019-01-05	2019-01-06		
Discount C	2019-01-01	2019-01-02	2019-01-04	2019-01-05			
List Price		2019-01-02		2019-01-05	2019-01-06	2019-01-08	

Figure 139 - Date Ranges Aggregation Rules Analysis 40

The final solution with the date ranges that contain a discount is:



Date Ranges Result

2019-01-02	2019-01-04	2019-01-05	2019-01-05	2019-01-06	2019-01-08

Figure 140 - Date Ranges Aggregation Rules - Analysis 40 Results

From the distinct Date Effective (2019-01-02) of discounts B and C to the Date Expired (2019-01-04) of discount C, discounts B and C are valid with the list price from 2019-01-02 to 2019-01-05. From the Date Effective (2019-01-05) of discount A to the Date Expired (2019-01-05) of discount B, discounts A and B are valid with the list price from 2019-01-02 to 2019-01-05. From the Date Expired (+1) (2019-01-06) of discount B to the Date Expired (2019-01-08) of discount A, discount A is valid with the list price from 2019-01-06 to 2019-01-08.

7.4.7. List Price Dates with 4 Discount Dates

In the same way as with 3 discount dates occurring at the same time of a List Price Date, with 4 discount dates, the List Price Date Effective and Date Expired scenarios don't call for the List Price dates to be introduced as the dates are already correctly separated into common date ranges.

When to Insert List Price Dates With 4 Discount Dates						
Analysis nr	**List Price**	**Discount A**	**Discount B**	**Discount C**	**Discount D**	**Action**
41	Date Effective	Date Effective (-1)	Date Effective	Date Expired	Date Expired (+1)	DON'T INSERT
42	Date Expired	Date Effective (-1)	Date Effective	Date Expired	Date Expired (+1)	DON'T INSERT

Table 32 - When to Insert List Price Dates With 4 Discount Dates

7.4.7.1. Analysis 41

In this scenario, the list price Date Effective is analyzed in the event that there's a discount Date Effective (-1), Date Effective, Date Expired and Date Expired (+1) occurring on the same date. In these cases, the list price Date Effective will never be considered in order to correctly segregate the date ranges.

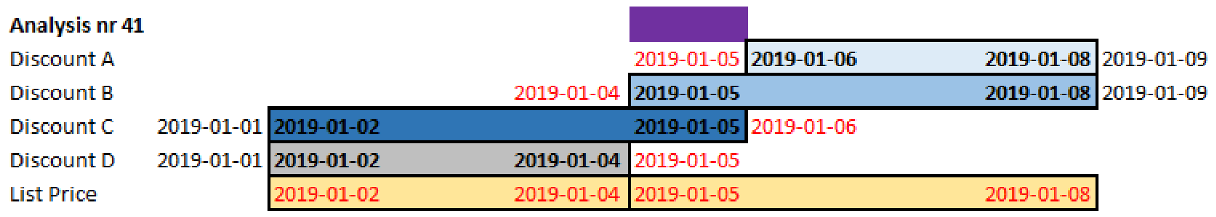


Figure 141 - Date Ranges Aggregation Rules Analysis 41

The final solution with the date ranges that contain a discount is:

Date Ranges Result

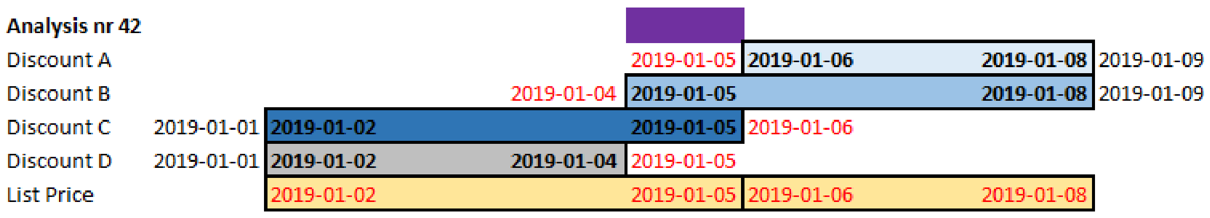
2019-01-02 2019-01-04	2019-01-05 2019-01-05	2019-01-06 2019-01-08

Figure 142 - Date Ranges Aggregation Rules - Analysis 41 Results

From the distinct Date Effective (2019-01-02) of discounts C and D to the Date Expired (2019-01-04) of discount D, discounts C and D are valid with the list price from 2019-01-02 to 2019-01-04. From the Date Effective (2019-01-05) of discount B to the Date Expired (2019-01-05) of discount C, discounts B and C are valid with the list price from 2019-01-05 to 2019-01-08. From the Date Effective (2019-01-06) of discount A to the distinct Date Expired (2019-01-08) of discounts A and B, discounts A and B are valid with the list price from 2019-01-05 to 2019-01-08.

7.4.7.2. Analysis 42

In this scenario, the list price Date Expired is analyzed in the event that there's a discount Date Effective (-1), Date Effective, Date Expired and Date Expired (+1) occurring on the same date. In these cases, the list price Date Expired will never be considered in order to correctly segregate the date ranges.



Analysis nr 42							
Discount A				2019-01-05	2019-01-06	2019-01-08	2019-01-09
Discount B			2019-01-04	2019-01-05		2019-01-08	2019-01-09
Discount C	2019-01-01	2019-01-02		2019-01-05	2019-01-06		
Discount D	2019-01-01	2019-01-02	2019-01-04	2019-01-05			
List Price		2019-01-02		2019-01-05	2019-01-06	2019-01-08	

Figure 143 - Date Ranges Aggregation Rules Analysis 42

The final solution with the date ranges that contain a discount is:

Date Ranges Result

2019-01-02 2019-01-04	2019-01-05 2019-01-05	2019-01-06 2019-01-08

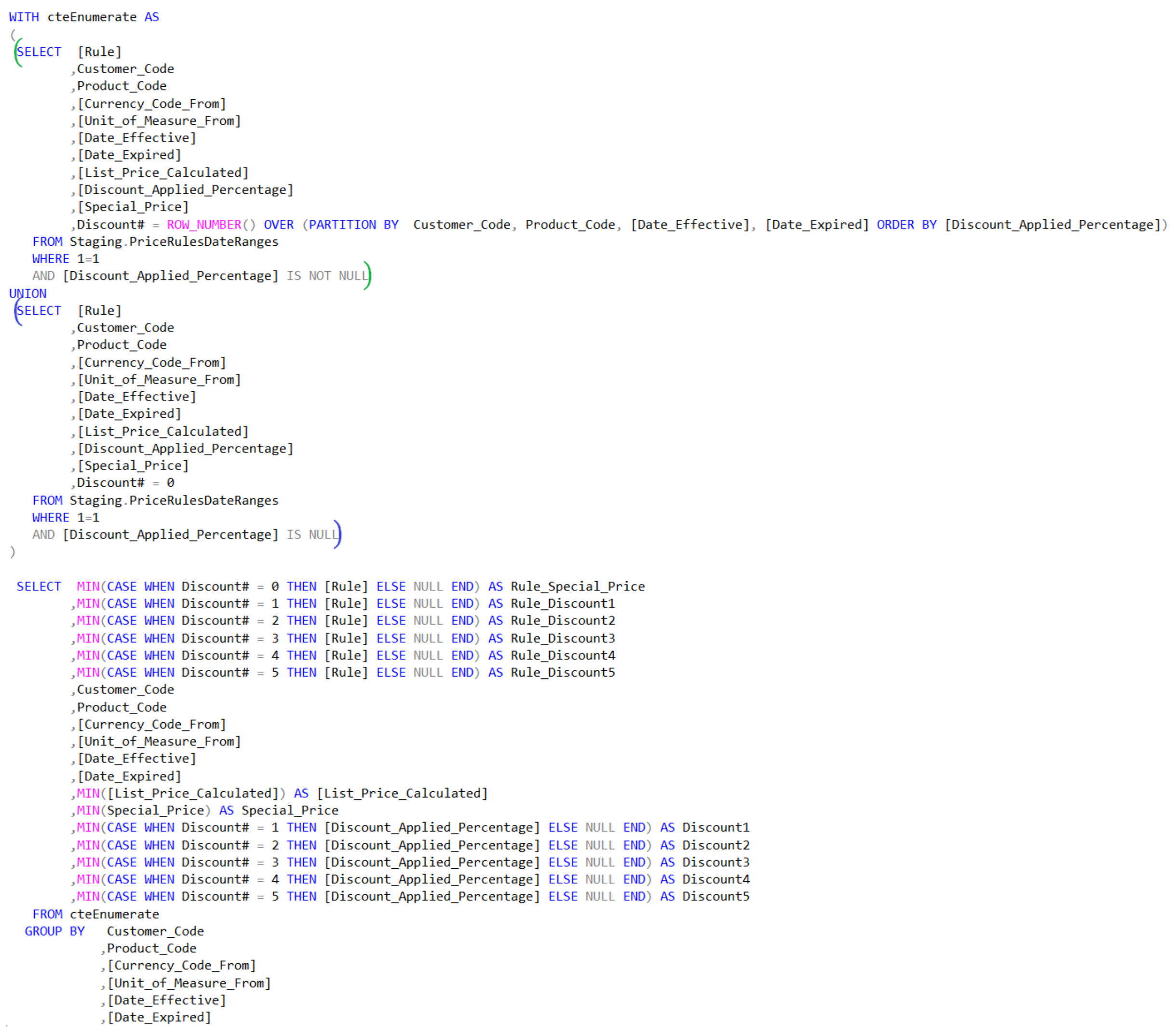
Figure 144 - Date Ranges Aggregation Rules - Analysis 42 Results

From the distinct Date Effective (2019-01-02) of discounts C and D to the Date Expired (2019-01-04) of discount D, discounts C and D are valid with the list price from 2019-01-02 to 2019-01-05. From the Date Effective (2019-01-05) of discount B to the Date Expired (2019-01-05) of discount C, discounts B and C are valid with the list price from 2019-01-02 to 2019-01-05. From the Date Effective (2019-01-06) of discount A to the distinct Date Expired (2019-01-08) of discounts A and B, discounts A and B are valid with the list price from 2019-01-06 to 2019-01-08.

7.5. PIVOT SSIS PACKAGE

7.5.1. Query - Pivot

The query used for the Pivot transformation phase is the following:

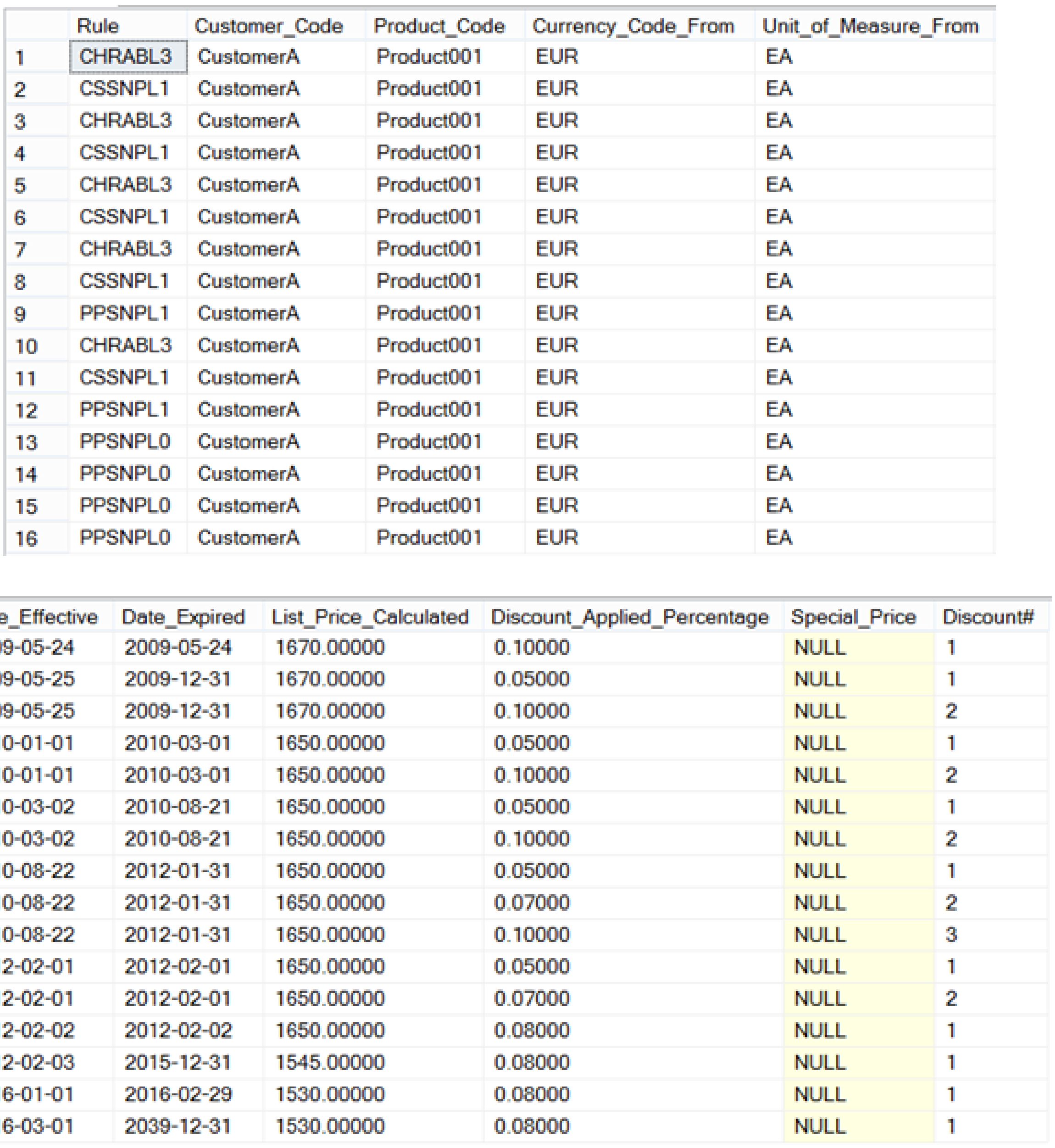


```sql
WITH cteEnumerate AS
(
(SELECT  [Rule]
        ,Customer_Code
        ,Product_Code
        ,[Currency_Code_From]
        ,[Unit_of_Measure_From]
        ,[Date_Effective]
        ,[Date_Expired]
        ,[List_Price_Calculated]
        ,[Discount_Applied_Percentage]
        ,[Special_Price]
        ,Discount# = ROW_NUMBER() OVER (PARTITION BY  Customer_Code, Product_Code, [Date_Effective], [Date_Expired] ORDER BY [Discount_Applied_Percentage])
   FROM Staging.PriceRulesDateRanges
   WHERE 1=1
   AND [Discount_Applied_Percentage] IS NOT NULL)
UNION
(SELECT  [Rule]
        ,Customer_Code
        ,Product_Code
        ,[Currency_Code_From]
        ,[Unit_of_Measure_From]
        ,[Date_Effective]
        ,[Date_Expired]
        ,[List_Price_Calculated]
        ,[Discount_Applied_Percentage]
        ,[Special_Price]
        ,Discount# = 0
   FROM Staging.PriceRulesDateRanges
   WHERE 1=1
   AND [Discount_Applied_Percentage] IS NULL)
)

 SELECT  MIN(CASE WHEN Discount# = 0 THEN [Rule] ELSE NULL END) AS Rule_Special_Price
        ,MIN(CASE WHEN Discount# = 1 THEN [Rule] ELSE NULL END) AS Rule_Discount1
        ,MIN(CASE WHEN Discount# = 2 THEN [Rule] ELSE NULL END) AS Rule_Discount2
        ,MIN(CASE WHEN Discount# = 3 THEN [Rule] ELSE NULL END) AS Rule_Discount3
        ,MIN(CASE WHEN Discount# = 4 THEN [Rule] ELSE NULL END) AS Rule_Discount4
        ,MIN(CASE WHEN Discount# = 5 THEN [Rule] ELSE NULL END) AS Rule_Discount5
        ,Customer_Code
        ,Product_Code
        ,[Currency_Code_From]
        ,[Unit_of_Measure_From]
        ,[Date_Effective]
        ,[Date_Expired]
        ,MIN([List_Price_Calculated]) AS [List_Price_Calculated]
        ,MIN(Special_Price) AS Special_Price
        ,MIN(CASE WHEN Discount# = 1 THEN [Discount_Applied_Percentage] ELSE NULL END) AS Discount1
        ,MIN(CASE WHEN Discount# = 2 THEN [Discount_Applied_Percentage] ELSE NULL END) AS Discount2
        ,MIN(CASE WHEN Discount# = 3 THEN [Discount_Applied_Percentage] ELSE NULL END) AS Discount3
        ,MIN(CASE WHEN Discount# = 4 THEN [Discount_Applied_Percentage] ELSE NULL END) AS Discount4
        ,MIN(CASE WHEN Discount# = 5 THEN [Discount_Applied_Percentage] ELSE NULL END) AS Discount5
   FROM cteEnumerate
  GROUP BY   Customer_Code
            ,Product_Code
            ,[Currency_Code_From]
            ,[Unit_of_Measure_From]
            ,[Date_Effective]
            ,[Date_Expired]
```

Figure 145 - Query Pivot

There are 2 major subqueries marked with green and blue square brackets, for explanation purposes.

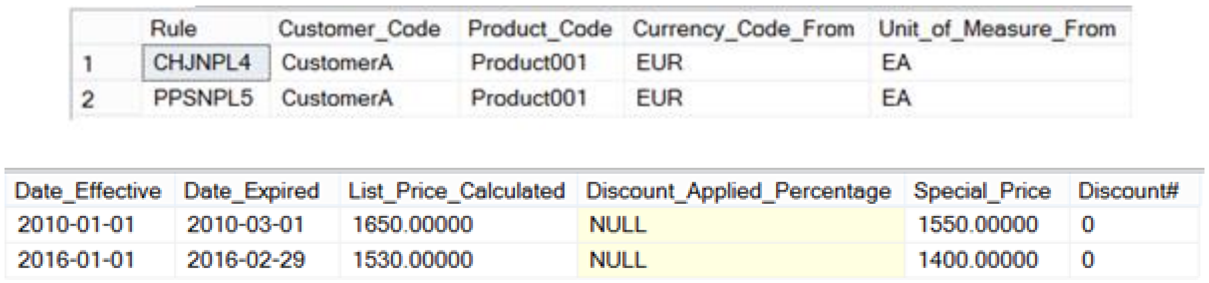
The green square bracket subquery takes all the cases where there is a discount in percentage and attributes a new column named Discount# that numbers every combination of customer_code, product_code, date_effective and date_expired with an ascending integer. Meaning that if there are 3 combinations of these same columns, there will be a number 1 for the lowest discount in percentage, a number 2 for the second lowest and a number 3 for the highest value. This is acceptable since the order in which the discounts in percentage are presented does not influence the calculation of the final price and also because the data regarding the order in which the rules are applied was not supplied by the client.



	Rule	Customer_Code	Product_Code	Currency_Code_From	Unit_of_Measure_From	Date_Effective	Date_Expired	List_Price_Calculated	Discount_Applied_Percentage	Special_Price	Discount#
1	CHRABL3	CustomerA	Product001	EUR	EA	2009-05-24	2009-05-24	1670.00000	0.10000	NULL	1
2	CSSNPL1	CustomerA	Product001	EUR	EA	2009-05-25	2009-12-31	1670.00000	0.05000	NULL	1
3	CHRABL3	CustomerA	Product001	EUR	EA	2009-05-25	2009-12-31	1670.00000	0.10000	NULL	2
4	CSSNPL1	CustomerA	Product001	EUR	EA	2010-01-01	2010-03-01	1650.00000	0.05000	NULL	1
5	CHRABL3	CustomerA	Product001	EUR	EA	2010-01-01	2010-03-01	1650.00000	0.10000	NULL	2
6	CSSNPL1	CustomerA	Product001	EUR	EA	2010-03-02	2010-08-21	1650.00000	0.05000	NULL	1
7	CHRABL3	CustomerA	Product001	EUR	EA	2010-03-02	2010-08-21	1650.00000	0.10000	NULL	2
8	CSSNPL1	CustomerA	Product001	EUR	EA	2010-08-22	2012-01-31	1650.00000	0.05000	NULL	1
9	PPSNPL1	CustomerA	Product001	EUR	EA	2010-08-22	2012-01-31	1650.00000	0.07000	NULL	2
10	CHRABL3	CustomerA	Product001	EUR	EA	2010-08-22	2012-01-31	1650.00000	0.10000	NULL	3
11	CSSNPL1	CustomerA	Product001	EUR	EA	2012-02-01	2012-02-01	1650.00000	0.05000	NULL	1
12	PPSNPL1	CustomerA	Product001	EUR	EA	2012-02-01	2012-02-01	1650.00000	0.07000	NULL	2
13	PPSNPL0	CustomerA	Product001	EUR	EA	2012-02-02	2012-02-02	1650.00000	0.08000	NULL	1
14	PPSNPL0	CustomerA	Product001	EUR	EA	2012-02-03	2015-12-31	1545.00000	0.08000	NULL	1
15	PPSNPL0	CustomerA	Product001	EUR	EA	2016-01-01	2016-02-29	1530.00000	0.08000	NULL	1
16	PPSNPL0	CustomerA	Product001	EUR	EA	2016-03-01	2039-12-31	1530.00000	0.08000	NULL	1

Table 33 - Subquery Pivot - Discounts in Percentage [Green Square Brackets]

The blue square bracket takes all the remaining special price records and attributes a Discount# column but with a 0 for all these records.



	Rule	Customer_Code	Product_Code	Currency_Code_From	Unit_of_Measure_From	Date_Effective	Date_Expired	List_Price_Calculated	Discount_Applied_Percentage	Special_Price	Discount#
1	CHJNPL4	CustomerA	Product001	EUR	EA	2010-01-01	2010-03-01	1650.00000	NULL	1550.00000	0
2	PPSNPL5	CustomerA	Product001	EUR	EA	2016-01-01	2016-02-29	1530.00000	NULL	1400.00000	0

Table 34 - Subquery Pivot - Special Prices [Blue Square Brackets]

Both the results are combined with a UNION ALL to be used as a CTE for the final transformation of the query. This final transformation replaces the original Rule column for the Rule_Special_Price and the 5 Rule_Discount columns. The same logic is done for the Discount_Applied_Percentage, which is replaced by the Discount1 through Discount5 columns. In essence, this enables multiple records for the same customer code, product code and date range to aggregate into one record while also presenting the discounts in percentage applied at the same time in an ascending order and their respective rules.

The query results are shown in Table 19 of the subchapter 3.3.2.3.

7.5.2. Final Price Calculated - SSIS Expression

The SSIS expression for the calculation of the final price is shown in Figure 39:

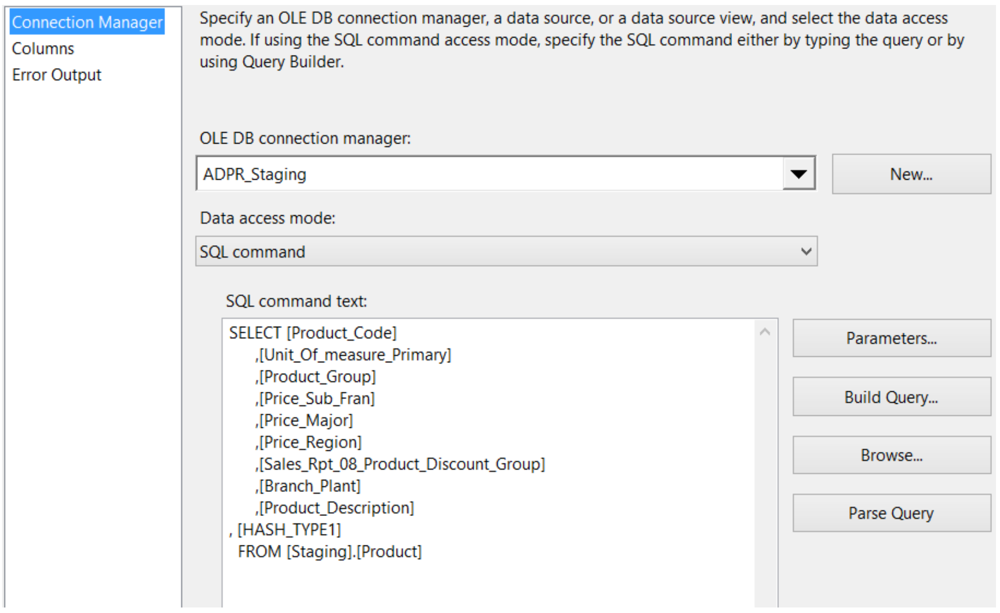
```
ISNULL((DT_DECIMAL,6)Special_Price) == FALSE && ISNULL((DT_DECIMAL,6)Discount1) == TRUE ? (DT_DECIMAL,6)Special_Price :

(ISNULL((DT_DECIMAL,6)Special_Price) == FALSE && ISNULL((DT_DECIMAL,6)Discount1) == FALSE && ISNULL((DT_DECIMAL,6)Discount2) == TRUE ?
(DT_DECIMAL,6)Special_Price * (1 - (DT_DECIMAL,6)Discount1) :

(ISNULL((DT_DECIMAL,6)Special_Price) == FALSE && ISNULL((DT_DECIMAL,6)Discount1) == FALSE && ISNULL((DT_DECIMAL,6)Discount2) == FALSE &&
ISNULL((DT_DECIMAL,6)Discount3) == TRUE ? (DT_DECIMAL,6)Special_Price * (1 - (DT_DECIMAL,6)Discount1) * (1 - (DT_DECIMAL,6)Discount2) :

(ISNULL((DT_DECIMAL,6)Special_Price) == FALSE && ISNULL((DT_DECIMAL,6)Discount1) == FALSE && ISNULL((DT_DECIMAL,6)Discount2) == FALSE &&
ISNULL((DT_DECIMAL,6)Discount3) == FALSE && ISNULL((DT_DECIMAL,6)Discount4) == TRUE ? (DT_DECIMAL,6)Special_Price * (1 - (DT_DECIMAL,6)Discount1) *
(1 - (DT_DECIMAL,6)Discount2) * (1 - (DT_DECIMAL,6)Discount3) :

(ISNULL((DT_DECIMAL,6)Special_Price) == FALSE && ISNULL((DT_DECIMAL,6)Discount1) == FALSE && ISNULL((DT_DECIMAL,6)Discount2) == FALSE &&
ISNULL((DT_DECIMAL,6)Discount3) == FALSE && ISNULL((DT_DECIMAL,6)Discount4) == FALSE && ISNULL((DT_DECIMAL,6)Discount5) == TRUE ?
(DT_DECIMAL,6)Special_Price * (1 - (DT_DECIMAL,6)Discount1) * (1 - (DT_DECIMAL,6)Discount2) * (1 - (DT_DECIMAL,6)Discount3) *
(1 - (DT_DECIMAL,6)Discount4) :

(ISNULL((DT_DECIMAL,6)Special_Price) == FALSE && ISNULL((DT_DECIMAL,6)Discount1) == FALSE && ISNULL((DT_DECIMAL,6)Discount2) == FALSE &&
ISNULL((DT_DECIMAL,6)Discount3) == FALSE && ISNULL((DT_DECIMAL,6)Discount4) == FALSE && ISNULL((DT_DECIMAL,6)Discount5) == FALSE ?
(DT_DECIMAL,6)Special_Price * (1 - (DT_DECIMAL,6)Discount1) * (1 - (DT_DECIMAL,6)Discount2) * (1 - (DT_DECIMAL,6)Discount3) *
(1 - (DT_DECIMAL,6)Discount4) * (1 - (DT_DECIMAL,6)Discount5) :

(ISNULL((DT_DECIMAL,6)Special_Price) == TRUE && ISNULL((DT_DECIMAL,6)Discount1) == TRUE ? (DT_DECIMAL,6)List_Price :

(ISNULL((DT_DECIMAL,6)Special_Price) == TRUE && ISNULL((DT_DECIMAL,6)Discount1) == FALSE && ISNULL((DT_DECIMAL,6)Discount2) == TRUE ?
(DT_DECIMAL,6)List_Price * (1 - (DT_DECIMAL,6)Discount1) :

(ISNULL((DT_DECIMAL,6)Special_Price) == TRUE && ISNULL((DT_DECIMAL,6)Discount1) == FALSE && ISNULL((DT DECIMAL,6)Discount2) == FALSE &&
ISNULL((DT_DECIMAL,6)Discount3) == TRUE ? (DT_DECIMAL,6)List_Price * (1 - (DT_DECIMAL,6)Discount1) * (1 - (DT_DECIMAL,6)Discount2) :

(ISNULL((DT_DECIMAL,6)Special_Price) == TRUE && ISNULL((DT_DECIMAL,6)Discount1) == FALSE && ISNULL((DT_DECIMAL,6)Discount2) == FALSE &&
ISNULL((DT_DECIMAL,6)Discount3) == FALSE && ISNULL((DT_DECIMAL,6)Discount4) == TRUE ? (DT_DECIMAL,6)List_Price *
(1 - (DT_DECIMAL,6)Discount1) * (1 - (DT_DECIMAL,6)Discount2) * (1 - (DT_DECIMAL,6)Discount3) :

(ISNULL((DT_DECIMAL,6)Special_Price) == TRUE && ISNULL((DT_DECIMAL,6)Discount1) == FALSE && ISNULL((DT_DECIMAL,6)Discount2) == FALSE &&
ISNULL((DT_DECIMAL,6)Discount3) == FALSE && ISNULL((DT_DECIMAL,6)Discount4) == FALSE && ISNULL((DT_DECIMAL,6)Discount5) == TRUE ?
(DT_DECIMAL,6)List_Price * (1 - (DT_DECIMAL,6)Discount1) * (1 - (DT_DECIMAL,6)Discount2) * (1 - (DT_DECIMAL,6)Discount3) *
(1 - (DT_DECIMAL,6)Discount4) :

(ISNULL((DT_DECIMAL,6)Special_Price) == TRUE && ISNULL((DT_DECIMAL,6)Discount1) == FALSE && ISNULL((DT_DECIMAL,6)Discount2) == FALSE &&
ISNULL((DT_DECIMAL,6)Discount3) == FALSE && ISNULL((DT_DECIMAL,6)Discount4) == FALSE && ISNULL((DT_DECIMAL,6)Discount5) == FALSE ?
(DT_DECIMAL,6)List_Price * (1 - (DT_DECIMAL,6)Discount1) * (1 - (DT_DECIMAL,6)Discount2) * (1 - (DT_DECIMAL,6)Discount3) *
(1 - (DT_DECIMAL,6)Discount4) * (1 - (DT_DECIMAL,6)Discount5) : 0)))))))))))
```

Figure 146 - SSIS Expression for Final_Price_Calculated

Note that the list_price is named differently from the unit_list_price in dwh.f_transactions because it is looked up in the data source and not a part of the original list price that is presented alongside the invoiced sales. Therefore, although they are supposedly conformed facts, they are labelled differently for data quality reasons.

7.6. Dimension Product SSIS Component Details

The component OLE_SRC - Staging Product is the source of data for the data flow. The columns listed are extracted from the staging table staging.product.



Specify an OLE DB connection manager, a data source, or a data source view, and select the data access mode. If using the SQL command access mode, specify the SQL command either by typing the query or by using Query Builder.

OLE DB connection manager: ADPR_Staging

Data access mode: SQL command

SQL command text:

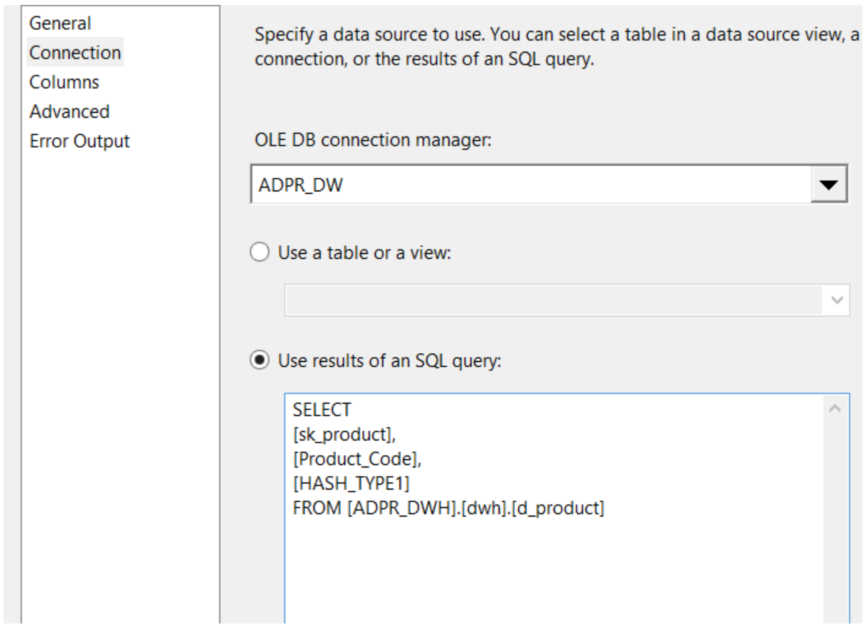
```
SELECT [Product_Code]
      ,[Unit_Of_measure_Primary]
      ,[Product_Group]
      ,[Price_Sub_Fran]
      ,[Price_Major]
      ,[Price_Region]
      ,[Sales_Rpt_08_Product_Discount_Group]
      ,[Branch_Plant]
      ,[Product_Description]
, [HASH_TYPE1]
  FROM [Staging].[Product]
```

Figure 147 - OLE_SRC - Staging Product Component

The column HASH_TYPE1 is comprised of a checksum function, which will formulate a unique code based on the combination of the columns that can be updated as part of the slowly-changing dimension type 1. The columns are the following:

Product_Description, Unit_Of_Measure_Primary, Product_Group, Price_Sub_Fran, Price_Major, Price_Region, Sales_Rpt_08_Product_Discount_Group, and Branch_Plant.

The next component is a lookup to the dimension table dwh.d_product in order to retrieve the surrogate key, the existing HASH_TYPE1 and the INSERT_DATE and UPDATE_DATE columns for logging purposes.



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

OLE DB connection manager:

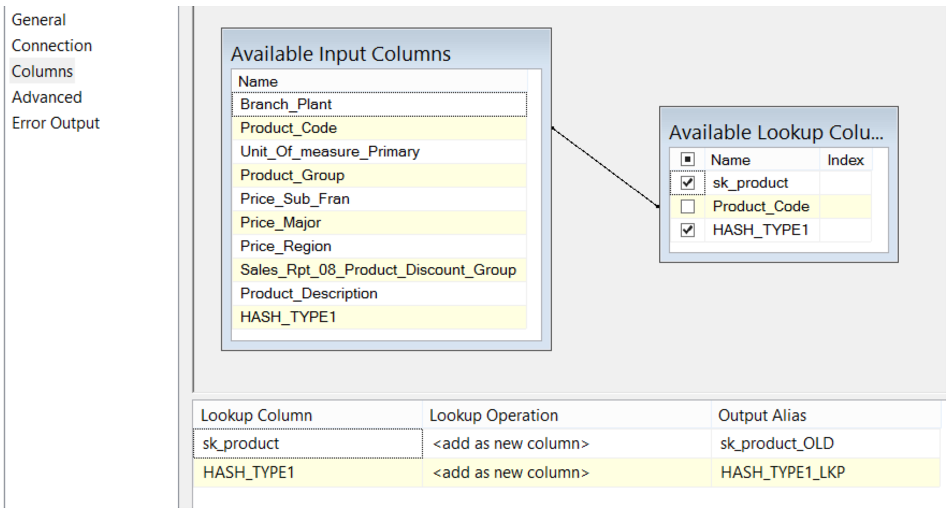
ADPR_DW

○ Use a table or a view:

● Use results of an SQL query:

```
SELECT
[sk_product],
[Product_Code],
[HASH_TYPE1]
FROM [ADPR_DWH].[dwh].[d_product]
```

Figure 148 – Connection Tab of the Lookup d_product Component



Available Input Columns

Name
Branch_Plant
Product_Code
Unit_Of_measure_Primary
Product_Group
Price_Sub_Fran
Price_Major
Price_Region
Sales_Rpt_08_Product_Discount_Group
Product_Description
HASH_TYPE1

Available Lookup Colu...

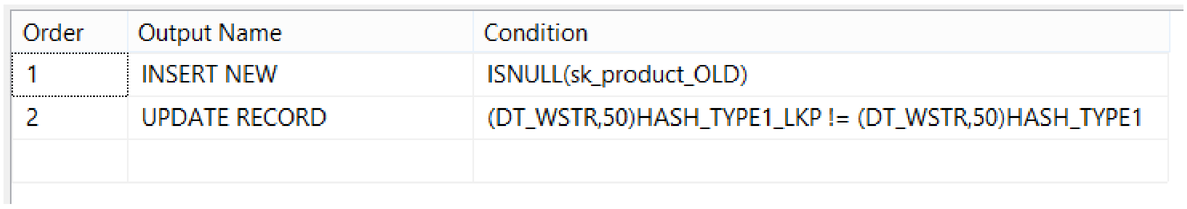
	Name	Index
✓	sk_product	
	Product_Code	
✓	HASH_TYPE1	

Lookup Column	Lookup Operation	Output Alias
sk_product	<add as new column>	sk_product_OLD
HASH_TYPE1	<add as new column>	HASH_TYPE1_LKP

Figure 149 - Columns Tab of the Lookup d_product Component

The Conditional Split component divides the data flow into two different flows. If the sk_product_OLD is null for a particular record, that means that the lookup for the product_code failed. This also means that the surrogate key from the dimension product doesn't exist for that product code, therefore, the record is new and is inserted.

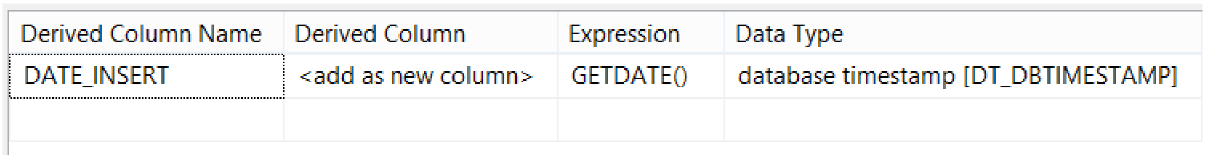
If the HASH_TYPE is different, it means that the record was able to lookup the product_code in the dimension, but at least one descriptive column changed.



Order	Output Name	Condition
1	INSERT NEW	ISNULL(sk_product_OLD)
2	UPDATE RECORD	(DT_WSTR,50)HASH_TYPE1_LKP != (DT_WSTR,50)HASH_TYPE1

Figure 150 - CSPL - UPDATE OR INSERT NEW RECORD Component

For the INSERT NEW flow, the derived column adds the column DATE_INSERT with a timestamp to register the time that the new records are inserted.



Derived Column Name	Derived Column	Expression	Data Type
DATE_INSERT	<add as new column>	GETDATE()	database timestamp [DT_DBTIMESTAMP]

Figure 151 - DER - INSERT_DATE Component

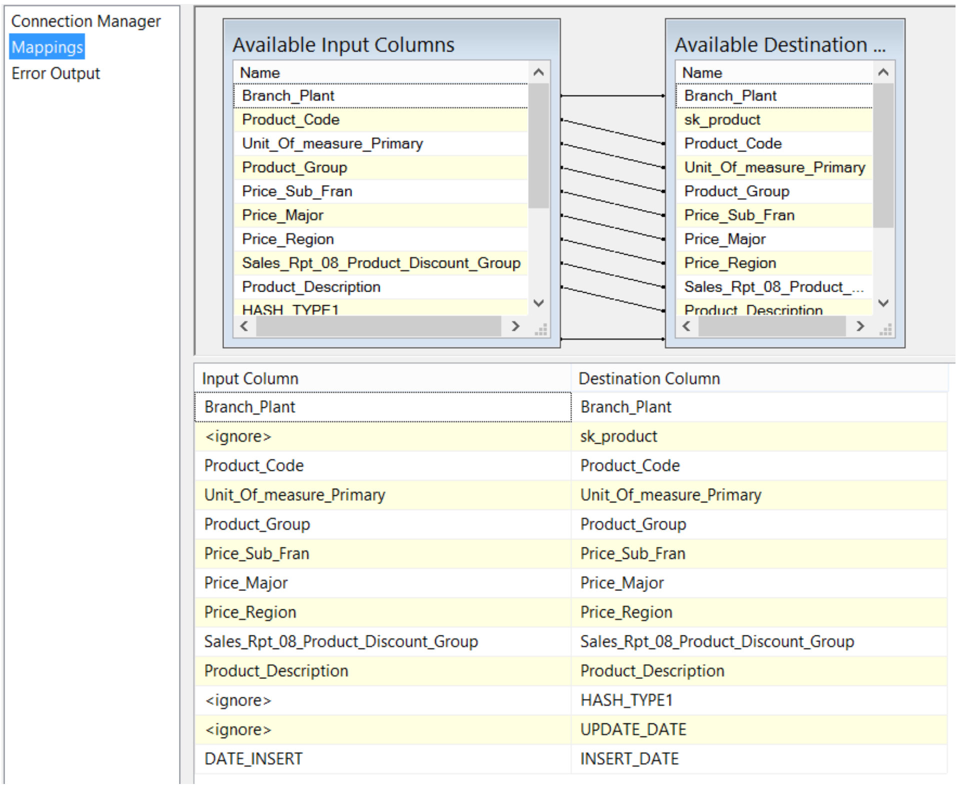
Before proceeding with the data flow, the dimension table is created. Note that the dimension also contains the column HASH_TYPE1, which is comprised of the same checksum function with the same combination of columns. Since this function is also present in the staging.product table, by comparing these 2 columns, it’s possible to verify if there have been any changes to records of the dimension that need to be overwritten.

The surrogate key is created using the IDENTITY(10,1) property, which attributes to each column, starting at number 10 and increasing by 1. The reason behind starting at number 10 is to leave room for the first 9 numbers to allocate inferred members such as unknowns, non-existent, and not applicable.

```
CREATE TABLE [dwh].[d_product](
    [sk_product] [int] IDENTITY(10,1) NOT NULL,
    [Product_Code] [nvarchar](255) NULL,
    [Product_Description] [nvarchar](255) NULL,
    [Unit_Of_measure_Primary] [nvarchar](255) NULL,
    [Product_Group] [nvarchar](255) NULL,
    [Price_Sub_Fran] [nvarchar](255) NULL,
    [Price_Major] [nvarchar](255) NULL,
    [Price_Region] [nvarchar](255) NULL,
    [Sales_Rpt_08_Product_Discount_Group] [nvarchar](255) NULL,
    [Branch_Plant] [nvarchar](255) NULL,
    [HASH_TYPE1]  AS (checksum([Product_Description],[Unit_Of_measure_Primary],[Product_Group],[Price_Sub_Fran],
                          [Price_Major],[Price_Region],[Sales_Rpt_08_Product_Discount_Group],[Branch_Plant])),
    [INSERT_DATE] [datetime] NULL,
    [UPDATE_DATE] [datetime] NULL,
 CONSTRAINT [PK_d_product] PRIMARY KEY CLUSTERED
(
    [sk_product] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]
```

Figure 152 - DDL Command to Create the Dimension Product

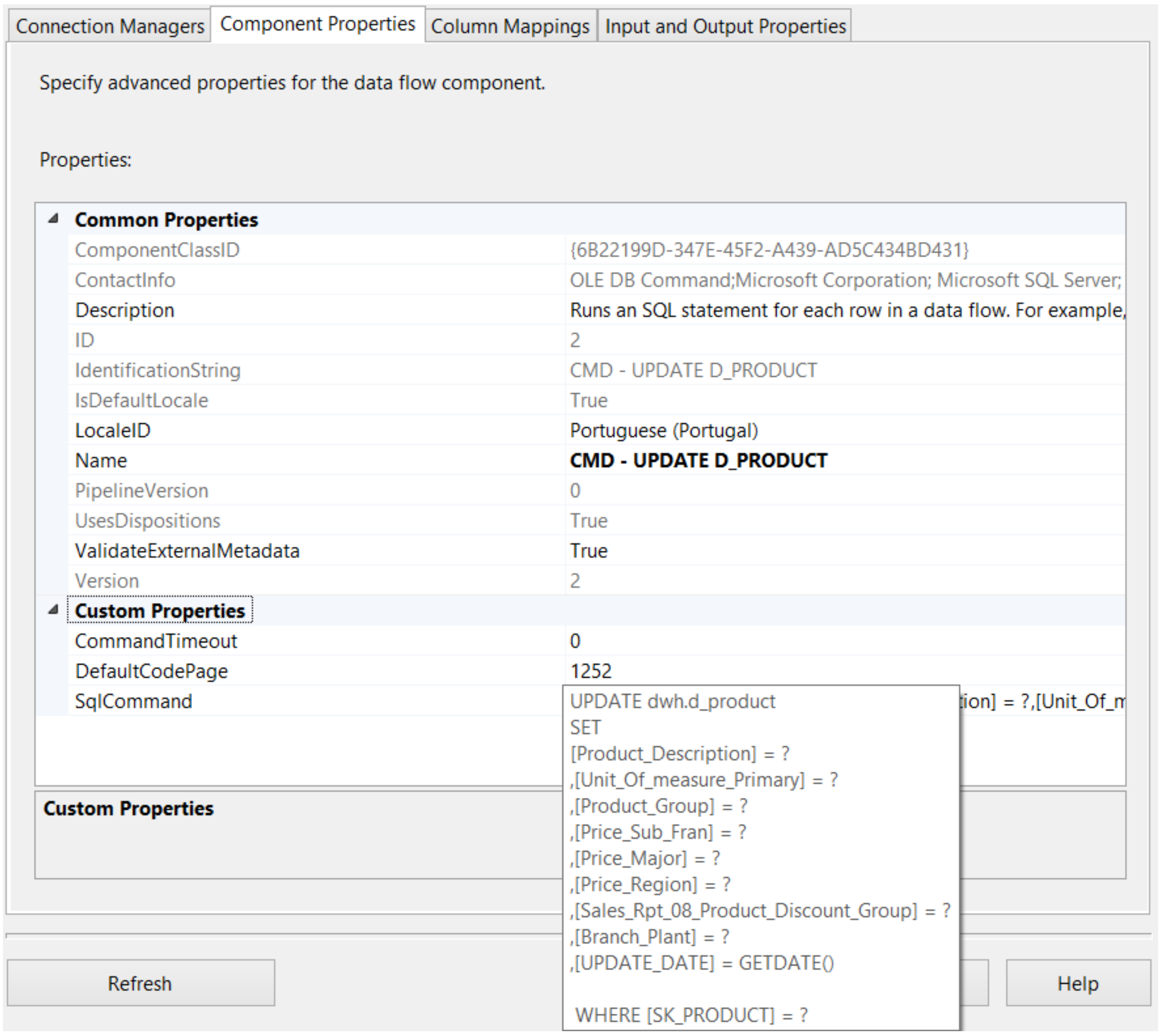
The destination component loads the data into the dimension product. The columns sk_product, HASH_TYPE1 and UPDATE_DATE are purposely left unmapped since the first two are populated with their functions and the third with the update command.



Input Column	Destination Column
Branch_Plant	Branch_Plant
<ignore>	sk_product
Product_Code	Product_Code
Unit_Of_measure_Primary	Unit_Of_measure_Primary
Product_Group	Product_Group
Price_Sub_Fran	Price_Sub_Fran
Price_Major	Price_Major
Price_Region	Price_Region
Sales_Rpt_08_Product_Discount_Group	Sales_Rpt_08_Product_Discount_Group
Product_Description	Product_Description
<ignore>	HASH_TYPE1
<ignore>	UPDATE_DATE
DATE_INSERT	INSERT_DATE

Figure 153 - OLE_DST - DWH Product Component

For the records that meet the update criteria from the condition split, the right flow is taken and the data is updated in the command component using the SQL command in Figure 154. All the descriptive columns are updated with the data that comes from the staging table and the UPDATE_DATE column is populated with the current date for logging purposes.



Specify advanced properties for the data flow component.

Properties:

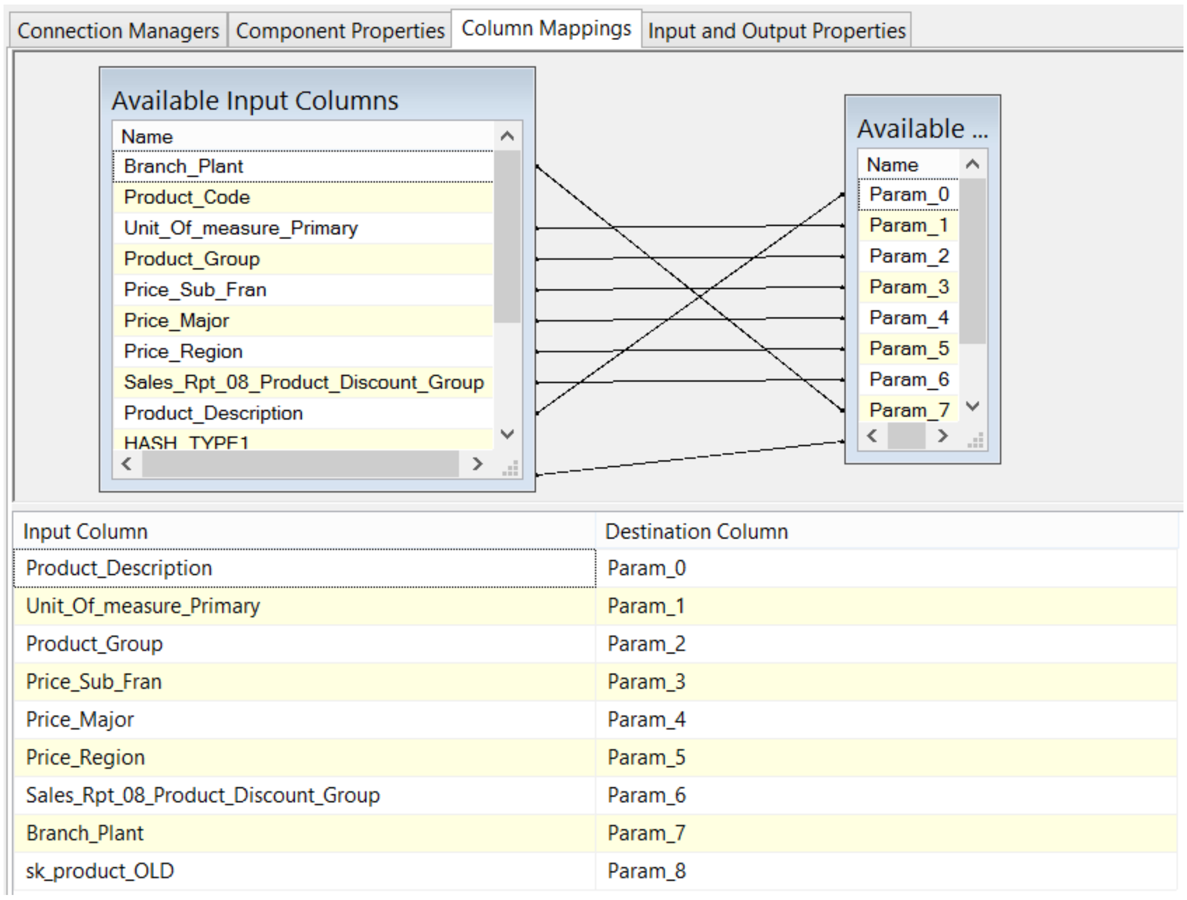
Common Properties	
ComponentClassID	{6B22199D-347E-45F2-A439-AD5C434BD431}
ContactInfo	OLE DB Command;Microsoft Corporation; Microsoft SQL Server;
Description	Runs an SQL statement for each row in a data flow. For example,
ID	2
IdentificationString	CMD - UPDATE D_PRODUCT
IsDefaultLocale	True
LocaleID	Portuguese (Portugal)
Name	**CMD - UPDATE D_PRODUCT**
PipelineVersion	0
UsesDispositions	True
ValidateExternalMetadata	True
Version	2
Custom Properties	
CommandTimeout	0
DefaultCodePage	1252
SqlCommand	ion] = ?,[Unit_Of_m

```
UPDATE dwh.d_product
SET
[Product_Description] = ?
,[Unit_Of_measure_Primary] = ?
,[Product_Group] = ?
,[Price_Sub_Fran] = ?
,[Price_Major] = ?
,[Price_Region] = ?
,[Sales_Rpt_08_Product_Discount_Group] = ?
,[Branch_Plant] = ?
,[UPDATE_DATE] = GETDATE()

 WHERE [SK_PRODUCT] = ?
```

Custom Properties

Figure 154 - CMD - UPDATE D_PRODUCT Component Properties

On the Column Mappings tab, each column from the SQL command in the Component Properties tab has to be ordered in the same way for each of the parameters, as shown in FIGURE 155.

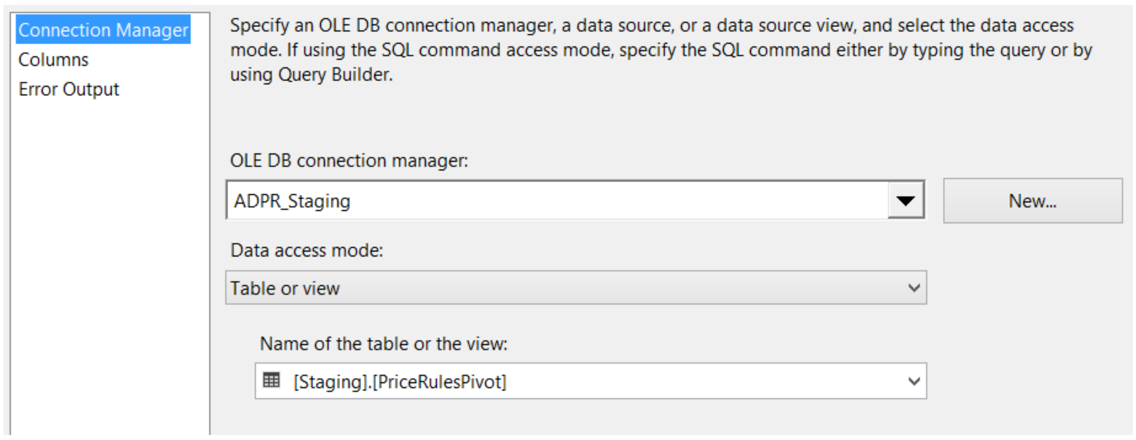


Input Column	Destination Column
Product_Description	Param_0
Unit_Of_measure_Primary	Param_1
Product_Group	Param_2
Price_Sub_Fran	Param_3
Price_Major	Param_4
Price_Region	Param_5
Sales_Rpt_08_Product_Discount_Group	Param_6
Branch_Plant	Param_7
sk_product_OLD	Param_8

Figure 155 - CMD - UPDATE D_PRODUCT Column Mappings

7.7. Fact Table Advanced Pricing SSIS Component Details

The data flow starts with the source component to extract the data from the staging.PriceRulesPivot.



Specify an OLE DB connection manager, a data source, or a data source view, and select the data access mode. If using the SQL command access mode, specify the SQL command either by typing the query or by using Query Builder.

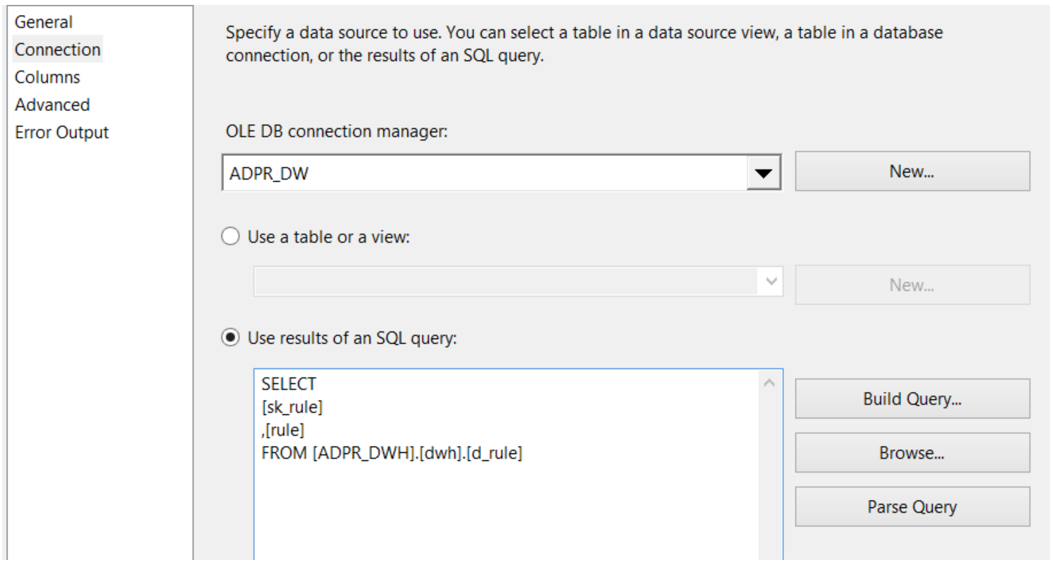
OLE DB connection manager: ADPR_Staging

Data access mode: Table or view

Name of the table or the view: [Staging].[PriceRulesPivot]

Figure 156 - OLE_SRC - Staging PriceRulesPivot Component

The lookup columns join the dimensions to obtain the respective surrogate key. Figures 157 and 158, illustrate the join, which is done between the Rule_Special_Price from the staging table and the column Rule from the dimension dwh.d_rule, in order to get the sk_rule_special_price_lkp, which will later be renamed to sk_rule_special_price.



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

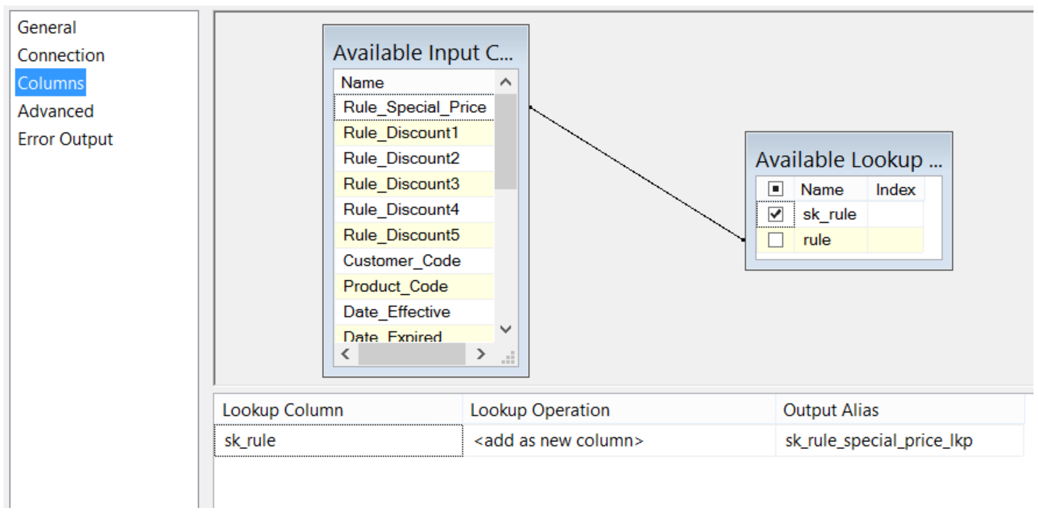
OLE DB connection manager: ADPR_DW

Use a table or a view:

[✓ Use results of an SQL query:]

```
SELECT
[sk_rule]
,[rule]
FROM [ADPR_DWH].[dwh].[d_rule]
```

Figure 157 - Lookup Rule Special Price - Component Connection

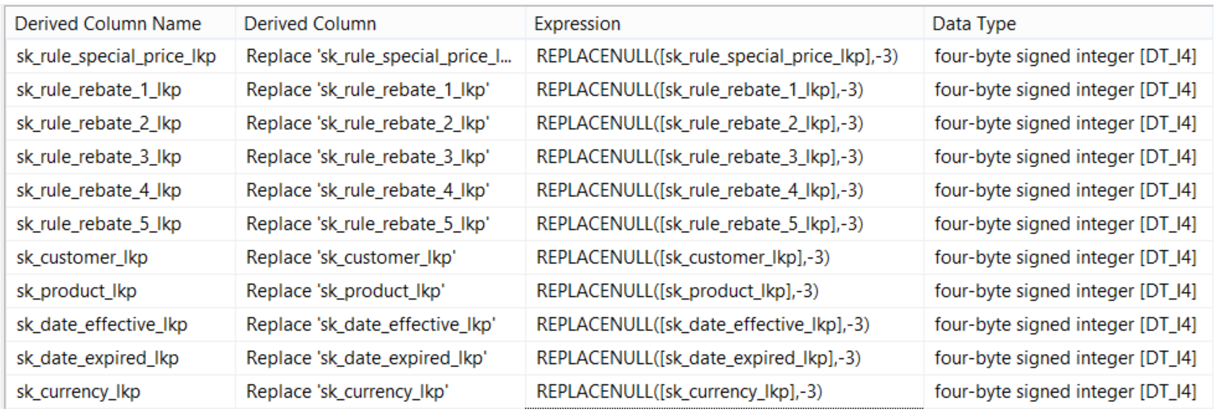


Lookup Column	Lookup Operation	Output Alias
sk_rule	<add as new column>	sk_rule_special_price_lkp

Figure 158 - Lookup Rule Special Price - Component Columns

After the lookups have finished, the multicast component splits the data into two data flows.

On the left data flow, the data is loaded into the fact table dwh.f_pricing, but must not have surrogate keys that are null, which are surrogate keys that failed the lookup to their respective dimension. These surrogate keys, if found null, are replaced by the inferred member -3, flagging the situation.



Derived Column Name	Derived Column	Expression	Data Type
sk_rule_special_price_lkp	Replace 'sk_rule_special_price_l...	REPLACENULL([sk_rule_special_price_lkp],-3)	four-byte signed integer [DT_I4]
sk_rule_rebate_1_lkp	Replace 'sk_rule_rebate_1_lkp'	REPLACENULL([sk_rule_rebate_1_lkp],-3)	four-byte signed integer [DT_I4]
sk_rule_rebate_2_lkp	Replace 'sk_rule_rebate_2_lkp'	REPLACENULL([sk_rule_rebate_2_lkp],-3)	four-byte signed integer [DT_I4]
sk_rule_rebate_3_lkp	Replace 'sk_rule_rebate_3_lkp'	REPLACENULL([sk_rule_rebate_3_lkp],-3)	four-byte signed integer [DT_I4]
sk_rule_rebate_4_lkp	Replace 'sk_rule_rebate_4_lkp'	REPLACENULL([sk_rule_rebate_4_lkp],-3)	four-byte signed integer [DT_I4]
sk_rule_rebate_5_lkp	Replace 'sk_rule_rebate_5_lkp'	REPLACENULL([sk_rule_rebate_5_lkp],-3)	four-byte signed integer [DT_I4]
sk_customer_lkp	Replace 'sk_customer_lkp'	REPLACENULL([sk_customer_lkp],-3)	four-byte signed integer [DT_I4]
sk_product_lkp	Replace 'sk_product_lkp'	REPLACENULL([sk_product_lkp],-3)	four-byte signed integer [DT_I4]
sk_date_effective_lkp	Replace 'sk_date_effective_lkp'	REPLACENULL([sk_date_effective_lkp],-3)	four-byte signed integer [DT_I4]
sk_date_expired_lkp	Replace 'sk_date_expired_lkp'	REPLACENULL([sk_date_expired_lkp],-3)	four-byte signed integer [DT_I4]
sk_currency_lkp	Replace 'sk_currency_lkp'	REPLACENULL([sk_currency_lkp],-3)	four-byte signed integer [DT_I4]

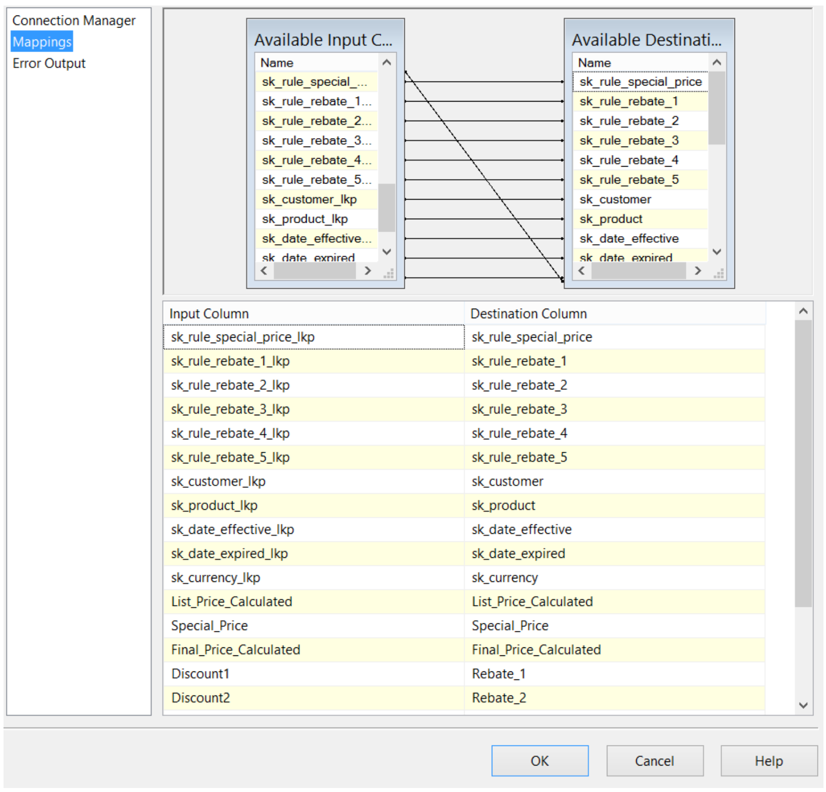
Figure 159 - DRC - Inferred Members - Component

Before loading the data into the fact table dwh.f_pricing, the table must first be created. The CREATE statement for the fact table is shown in Figure 160.

```
CREATE TABLE [dwh].[f_pricing](
    [sk_rule_special_price] [int] NOT NULL,
    [sk_rule_rebate_1] [int] NOT NULL,
    [sk_rule_rebate_2] [int] NOT NULL,
    [sk_rule_rebate_3] [int] NOT NULL,
    [sk_rule_rebate_4] [int] NOT NULL,
    [sk_rule_rebate_5] [int] NOT NULL,
    [sk_customer] [int] NOT NULL,
    [sk_product] [int] NOT NULL,
    [sk_date_effective] [int] NOT NULL,
    [sk_date_expired] [int] NOT NULL,
    [sk_currency] [int] NOT NULL,
    [List_Price_Calculated] [numeric](18, 5) NULL,
    [Special_Price] [numeric](18, 5) NULL,
    [Rebate_1] [numeric](18, 5) NULL,
    [Rebate_2] [numeric](18, 5) NULL,
    [Rebate_3] [numeric](18, 5) NULL,
    [Rebate_4] [numeric](18, 5) NULL,
    [Rebate_5] [numeric](18, 5) NULL,
    [Final_Price_Calculated] [numeric](38, 6) NULL,
    [Has_Special_Price] [nvarchar](50) NULL,
    [Number_Of_Discounts] [int] NULL
) ON [PRIMARY]
```

Figure 160 - CREATE Statement for dwh.f_pricing

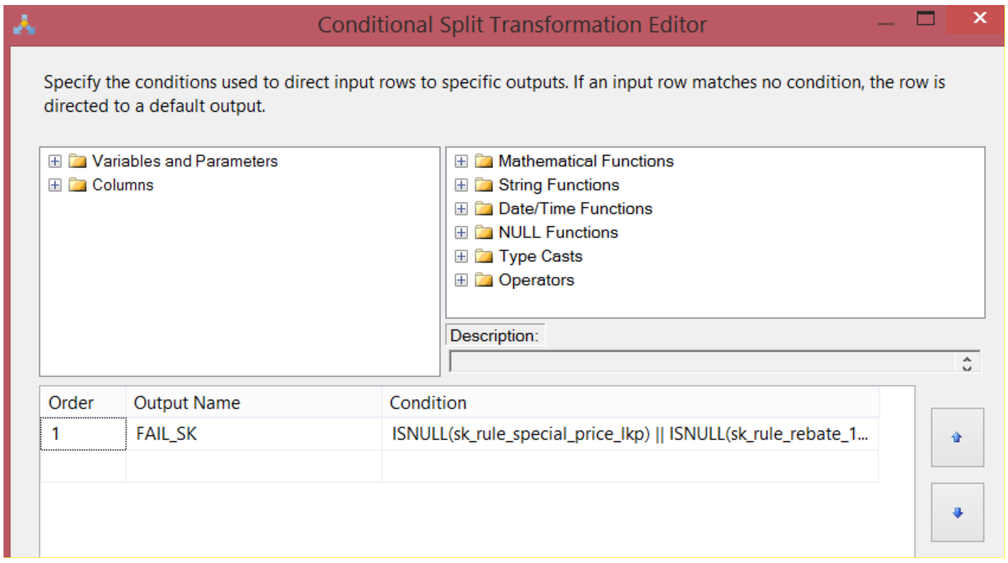
After the derived column has finished checking the data for null surrogate keys, it's then populated into the fact table dwh.f_pricing. The surrogate keys are also appropriately renamed.



Input Column	Destination Column
sk_rule_special_price_lkp	sk_rule_special_price
sk_rule_rebate_1_lkp	sk_rule_rebate_1
sk_rule_rebate_2_lkp	sk_rule_rebate_2
sk_rule_rebate_3_lkp	sk_rule_rebate_3
sk_rule_rebate_4_lkp	sk_rule_rebate_4
sk_rule_rebate_5_lkp	sk_rule_rebate_5
sk_customer_lkp	sk_customer
sk_product_lkp	sk_product
sk_date_effective_lkp	sk_date_effective
sk_date_expired_lkp	sk_date_expired
sk_currency_lkp	sk_currency
List_Price_Calculated	List_Price_Calculated
Special_Price	Special_Price
Final_Price_Calculated	Final_Price_Calculated
Discount1	Rebate_1
Discount2	Rebate_2

Figure 161 - OLE_DST - DWH f_pricing - Component

On the right data flow, a conditional split is used to filter the records that have at least one surrogate key that is null using the logical OR (||) in the expression, directing these records to a FAIL_SK output.


Conditional Split Transformation Editor

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

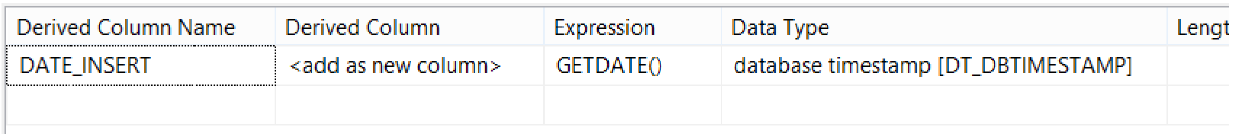
Order	Output Name	Condition
1	FAIL_SK	ISNULL(sk_rule_special_price_lkp) || ISNULL(sk_rule_rebate_1...

Figure 162 - Conditional Split - Component

```
ISNULL(sk_rule_special_price_lkp) || ISNULL(sk_rule_rebate_1_lkp) || ISNULL(sk_rule_rebate_2_lkp) ||
ISNULL(sk_rule_rebate_3_lkp) || ISNULL(sk_rule_rebate_4_lkp) || ISNULL(sk_rule_rebate_5_lkp) ||
ISNULL(sk_customer_lkp) || ISNULL(sk_product_lkp) || ISNULL(sk_date_effective_lkp) ||
ISNULL(sk_date_expired_lkp) || ISNULL(sk_currency_lkp)
```

Figure 163 - Conditional Split - Full Expression

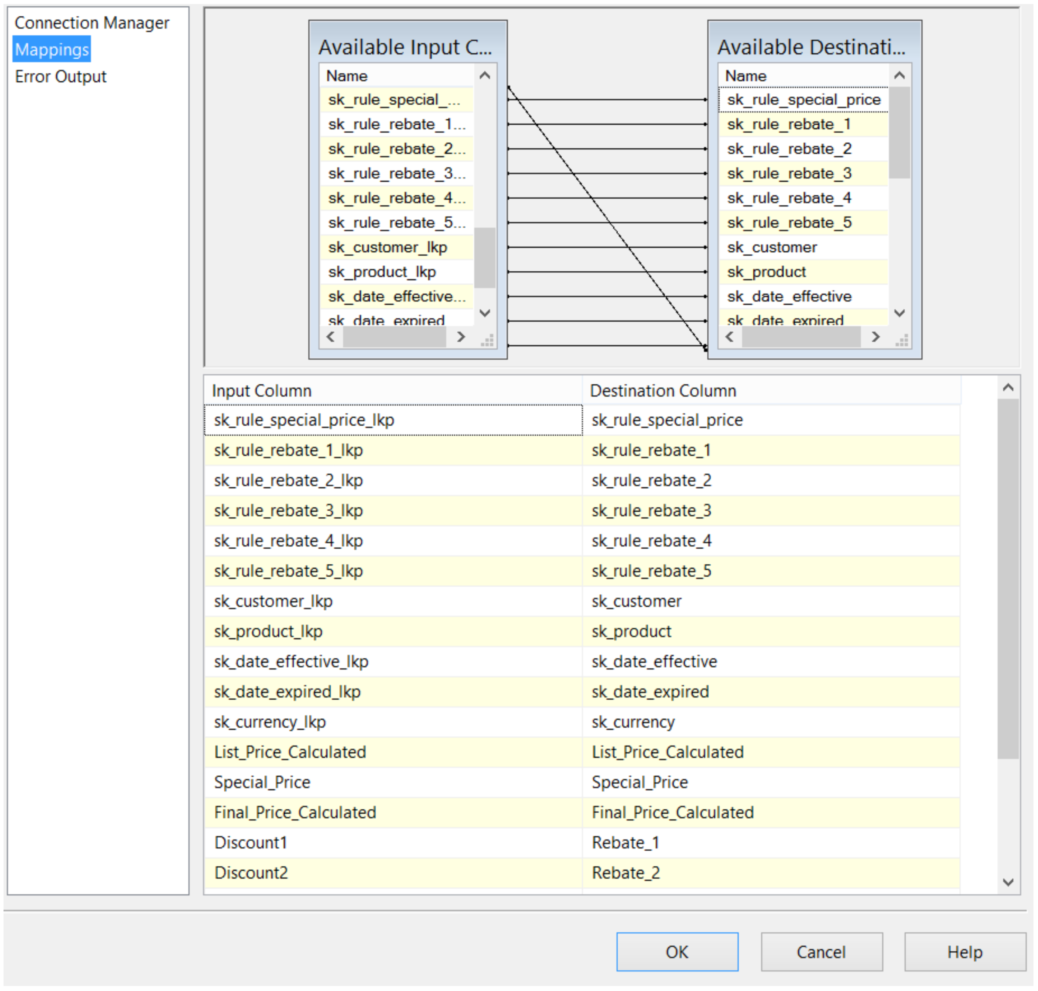
An insert date is then added to these records to give visibility of which day they were loaded.



Derived Column Name	Derived Column	Expression	Data Type	Lengt
DATE_INSERT	<add as new column>	GETDATE()	database timestamp [DT_DBTIMESTAMP]	

Figure 164 - DER - INSERT_DATE - Component

The log table has an identical structure of the fact table dwh.f_pricing to show all the columns that refer to the records that have at least one null surrogate key.

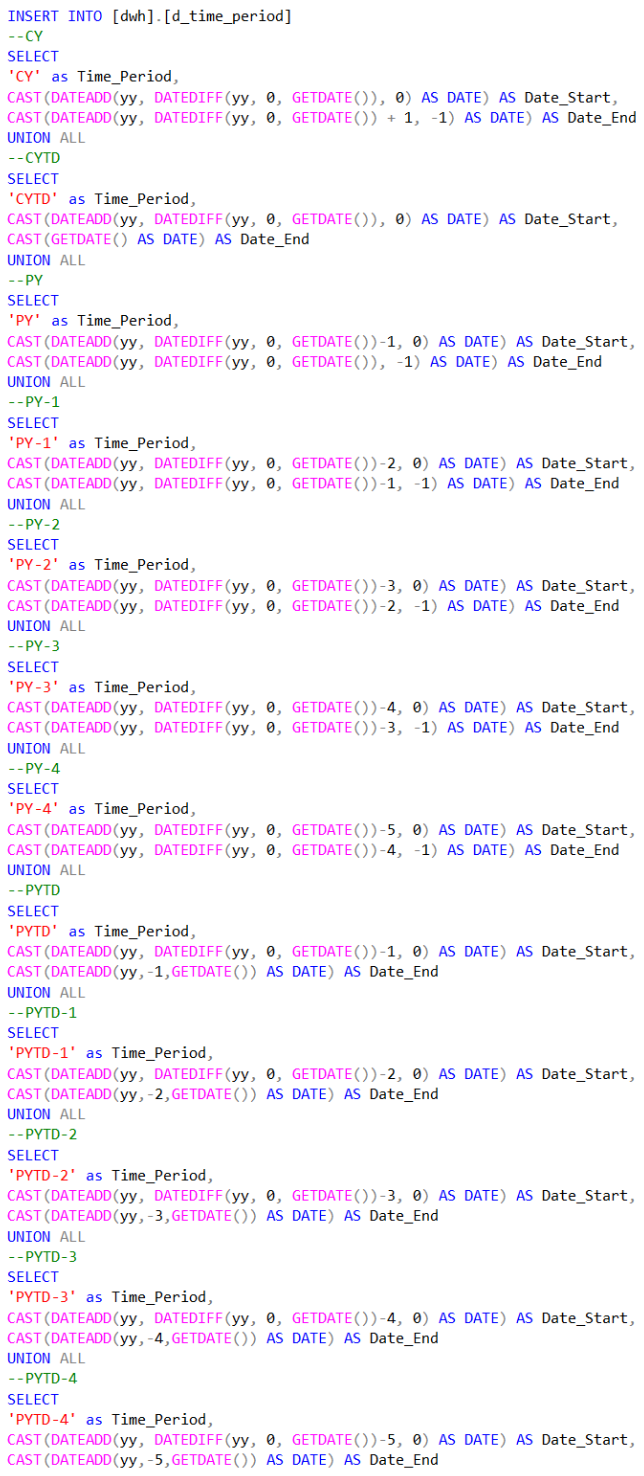


Input Column	Destination Column
sk_rule_special_price_lkp	sk_rule_special_price
sk_rule_rebate_1_lkp	sk_rule_rebate_1
sk_rule_rebate_2_lkp	sk_rule_rebate_2
sk_rule_rebate_3_lkp	sk_rule_rebate_3
sk_rule_rebate_4_lkp	sk_rule_rebate_4
sk_rule_rebate_5_lkp	sk_rule_rebate_5
sk_customer_lkp	sk_customer
sk_product_lkp	sk_product
sk_date_effective_lkp	sk_date_effective
sk_date_expired_lkp	sk_date_expired
sk_currency_lkp	sk_currency
List_Price_Calculated	List_Price_Calculated
Special_Price	Special_Price
Final_Price_Calculated	Final_Price_Calculated
Discount1	Rebate_1
Discount2	Rebate_2

Figure 165 - OLE_DST - DWH f_pricing_no_sk_match - Component

7.8. Qlik Sense Time Period

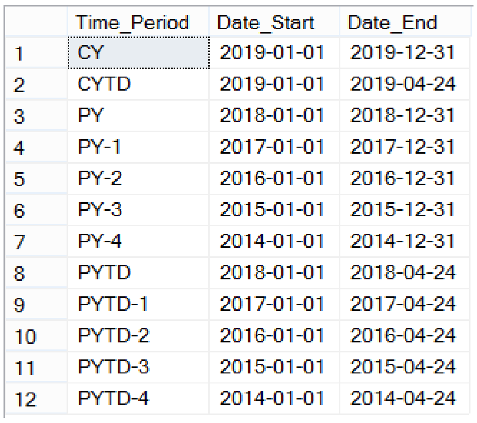
In the event that a transaction-oriented app is created for other purposes such as sales analysis, the introduction of time periods may add value to the dashboards. As such, another dimension was created that would need to be daily refreshed, contrary to the main project app that consists of the combination of transactions with advanced pricing agreements. The query to insert the data in this dimension every day is shown in Figure 166.



```sql
INSERT INTO [dwh].[d_time_period]
--CY
SELECT
'CY' as Time_Period,
CAST(DATEADD(yy, DATEDIFF(yy, 0, GETDATE()), 0) AS DATE) AS Date_Start,
CAST(DATEADD(yy, DATEDIFF(yy, 0, GETDATE()) + 1, -1) AS DATE) AS Date_End
UNION ALL
--CYTD
SELECT
'CYTD' as Time_Period,
CAST(DATEADD(yy, DATEDIFF(yy, 0, GETDATE()), 0) AS DATE) AS Date_Start,
CAST(GETDATE() AS DATE) AS Date_End
UNION ALL
--PY
SELECT
'PY' as Time_Period,
CAST(DATEADD(yy, DATEDIFF(yy, 0, GETDATE())-1, 0) AS DATE) AS Date_Start,
CAST(DATEADD(yy, DATEDIFF(yy, 0, GETDATE()), -1) AS DATE) AS Date_End
UNION ALL
--PY-1
SELECT
'PY-1' as Time_Period,
CAST(DATEADD(yy, DATEDIFF(yy, 0, GETDATE())-2, 0) AS DATE) AS Date_Start,
CAST(DATEADD(yy, DATEDIFF(yy, 0, GETDATE())-1, -1) AS DATE) AS Date_End
UNION ALL
--PY-2
SELECT
'PY-2' as Time_Period,
CAST(DATEADD(yy, DATEDIFF(yy, 0, GETDATE())-3, 0) AS DATE) AS Date_Start,
CAST(DATEADD(yy, DATEDIFF(yy, 0, GETDATE())-2, -1) AS DATE) AS Date_End
UNION ALL
--PY-3
SELECT
'PY-3' as Time_Period,
CAST(DATEADD(yy, DATEDIFF(yy, 0, GETDATE())-4, 0) AS DATE) AS Date_Start,
CAST(DATEADD(yy, DATEDIFF(yy, 0, GETDATE())-3, -1) AS DATE) AS Date_End
UNION ALL
--PY-4
SELECT
'PY-4' as Time_Period,
CAST(DATEADD(yy, DATEDIFF(yy, 0, GETDATE())-5, 0) AS DATE) AS Date_Start,
CAST(DATEADD(yy, DATEDIFF(yy, 0, GETDATE())-4, -1) AS DATE) AS Date_End
UNION ALL
--PYTD
SELECT
'PYTD' as Time_Period,
CAST(DATEADD(yy, DATEDIFF(yy, 0, GETDATE())-1, 0) AS DATE) AS Date_Start,
CAST(DATEADD(yy,-1,GETDATE()) AS DATE) AS Date_End
UNION ALL
--PYTD-1
SELECT
'PYTD-1' as Time_Period,
CAST(DATEADD(yy, DATEDIFF(yy, 0, GETDATE())-2, 0) AS DATE) AS Date_Start,
CAST(DATEADD(yy,-2,GETDATE()) AS DATE) AS Date_End
UNION ALL
--PYTD-2
SELECT
'PYTD-2' as Time_Period,
CAST(DATEADD(yy, DATEDIFF(yy, 0, GETDATE())-3, 0) AS DATE) AS Date_Start,
CAST(DATEADD(yy,-3,GETDATE()) AS DATE) AS Date_End
UNION ALL
--PYTD-3
SELECT
'PYTD-3' as Time_Period,
CAST(DATEADD(yy, DATEDIFF(yy, 0, GETDATE())-4, 0) AS DATE) AS Date_Start,
CAST(DATEADD(yy,-4,GETDATE()) AS DATE) AS Date_End
UNION ALL
--PYTD-4
SELECT
'PYTD-4' as Time_Period,
CAST(DATEADD(yy, DATEDIFF(yy, 0, GETDATE())-5, 0) AS DATE) AS Date_Start,
CAST(DATEADD(yy,-5,GETDATE()) AS DATE) AS Date_End
```

Figure 166 - Query dwh.d_time_period

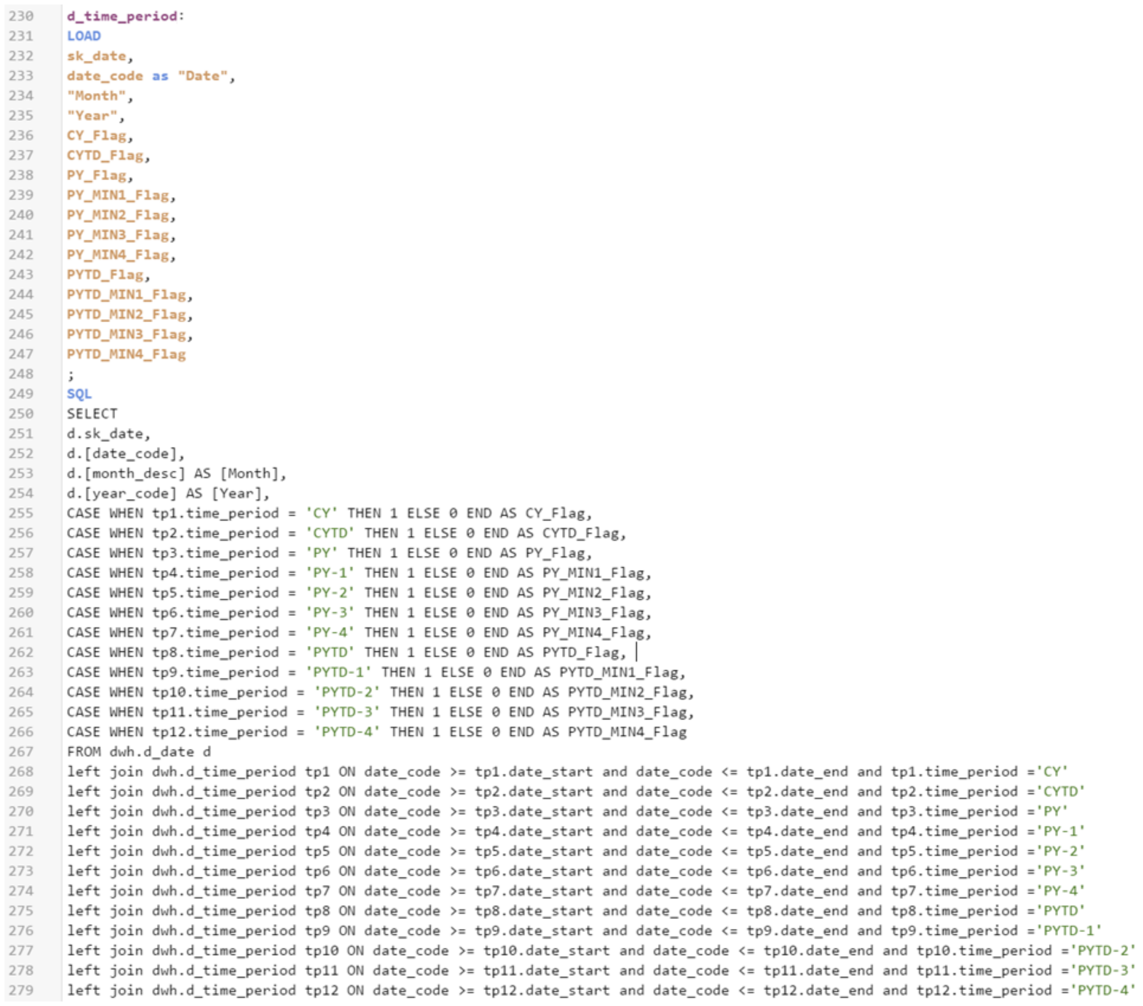
By executing this query and selecting the table, the results show the various time periods and the corresponding start and end dates:



	Time_Period	Date_Start	Date_End
1	CY	2019-01-01	2019-12-31
2	CYTD	2019-01-01	2019-04-24
3	PY	2018-01-01	2018-12-31
4	PY-1	2017-01-01	2017-12-31
5	PY-2	2016-01-01	2016-12-31
6	PY-3	2015-01-01	2015-12-31
7	PY-4	2014-01-01	2014-12-31
8	PYTD	2018-01-01	2018-04-24
9	PYTD-1	2017-01-01	2017-04-24
10	PYTD-2	2016-01-01	2016-04-24
11	PYTD-3	2015-01-01	2015-04-24
12	PYTD-4	2014-01-01	2014-04-24

Table 35 - dwh.d_time_period results

On the Qlik Sense, the data load must include a left join from the Date dimension to the Time Period dimension for each time period in order to separate the attributes by those same periods.



```
d_time_period:
LOAD
sk_date,
date_code as "Date",
"Month",
"Year",
CY_Flag,
CYTD_Flag,
PY_Flag,
PY_MIN1_Flag,
PY_MIN2_Flag,
PY_MIN3_Flag,
PY_MIN4_Flag,
PYTD_Flag,
PYTD_MIN1_Flag,
PYTD_MIN2_Flag,
PYTD_MIN3_Flag,
PYTD_MIN4_Flag
;
SQL
SELECT
d.sk_date,
d.[date_code],
d.[month_desc] AS [Month],
d.[year_code] AS [Year],
CASE WHEN tp1.time_period = 'CY' THEN 1 ELSE 0 END AS CY_Flag,
CASE WHEN tp2.time_period = 'CYTD' THEN 1 ELSE 0 END AS CYTD_Flag,
CASE WHEN tp3.time_period = 'PY' THEN 1 ELSE 0 END AS PY_Flag,
CASE WHEN tp4.time_period = 'PY-1' THEN 1 ELSE 0 END AS PY_MIN1_Flag,
CASE WHEN tp5.time_period = 'PY-2' THEN 1 ELSE 0 END AS PY_MIN2_Flag,
CASE WHEN tp6.time_period = 'PY-3' THEN 1 ELSE 0 END AS PY_MIN3_Flag,
CASE WHEN tp7.time_period = 'PY-4' THEN 1 ELSE 0 END AS PY_MIN4_Flag,
CASE WHEN tp8.time_period = 'PYTD' THEN 1 ELSE 0 END AS PYTD_Flag,
CASE WHEN tp9.time_period = 'PYTD-1' THEN 1 ELSE 0 END AS PYTD_MIN1_Flag,
CASE WHEN tp10.time_period = 'PYTD-2' THEN 1 ELSE 0 END AS PYTD_MIN2_Flag,
CASE WHEN tp11.time_period = 'PYTD-3' THEN 1 ELSE 0 END AS PYTD_MIN3_Flag,
CASE WHEN tp12.time_period = 'PYTD-4' THEN 1 ELSE 0 END AS PYTD_MIN4_Flag
FROM dwh.d_date d
left join dwh.d_time_period tp1 ON date_code >= tp1.date_start and date_code <= tp1.date_end and tp1.time_period ='CY'
left join dwh.d_time_period tp2 ON date_code >= tp2.date_start and date_code <= tp2.date_end and tp2.time_period ='CYTD'
left join dwh.d_time_period tp3 ON date_code >= tp3.date_start and date_code <= tp3.date_end and tp3.time_period ='PY'
left join dwh.d_time_period tp4 ON date_code >= tp4.date_start and date_code <= tp4.date_end and tp4.time_period ='PY-1'
left join dwh.d_time_period tp5 ON date_code >= tp5.date_start and date_code <= tp5.date_end and tp5.time_period ='PY-2'
left join dwh.d_time_period tp6 ON date_code >= tp6.date_start and date_code <= tp6.date_end and tp6.time_period ='PY-3'
left join dwh.d_time_period tp7 ON date_code >= tp7.date_start and date_code <= tp7.date_end and tp7.time_period ='PY-4'
left join dwh.d_time_period tp8 ON date_code >= tp8.date_start and date_code <= tp8.date_end and tp8.time_period ='PYTD'
left join dwh.d_time_period tp9 ON date_code >= tp9.date_start and date_code <= tp9.date_end and tp9.time_period ='PYTD-1'
left join dwh.d_time_period tp10 ON date_code >= tp10.date_start and date_code <= tp10.date_end and tp10.time_period ='PYTD-2'
left join dwh.d_time_period tp11 ON date_code >= tp11.date_start and date_code <= tp11.date_end and tp11.time_period ='PYTD-3'
left join dwh.d_time_period tp12 ON date_code >= tp12.date_start and date_code <= tp12.date_end and tp12.time_period ='PYTD-4'
```

Figure 167 - d_time_period

Although this enhanced time dimension would better serve a Sales Analysis app, if it were introduced to the main app, the data model would have the Date dimension in the data warehouse, named Time dimension in Qlik, would be swapped by the Time Period dimension:

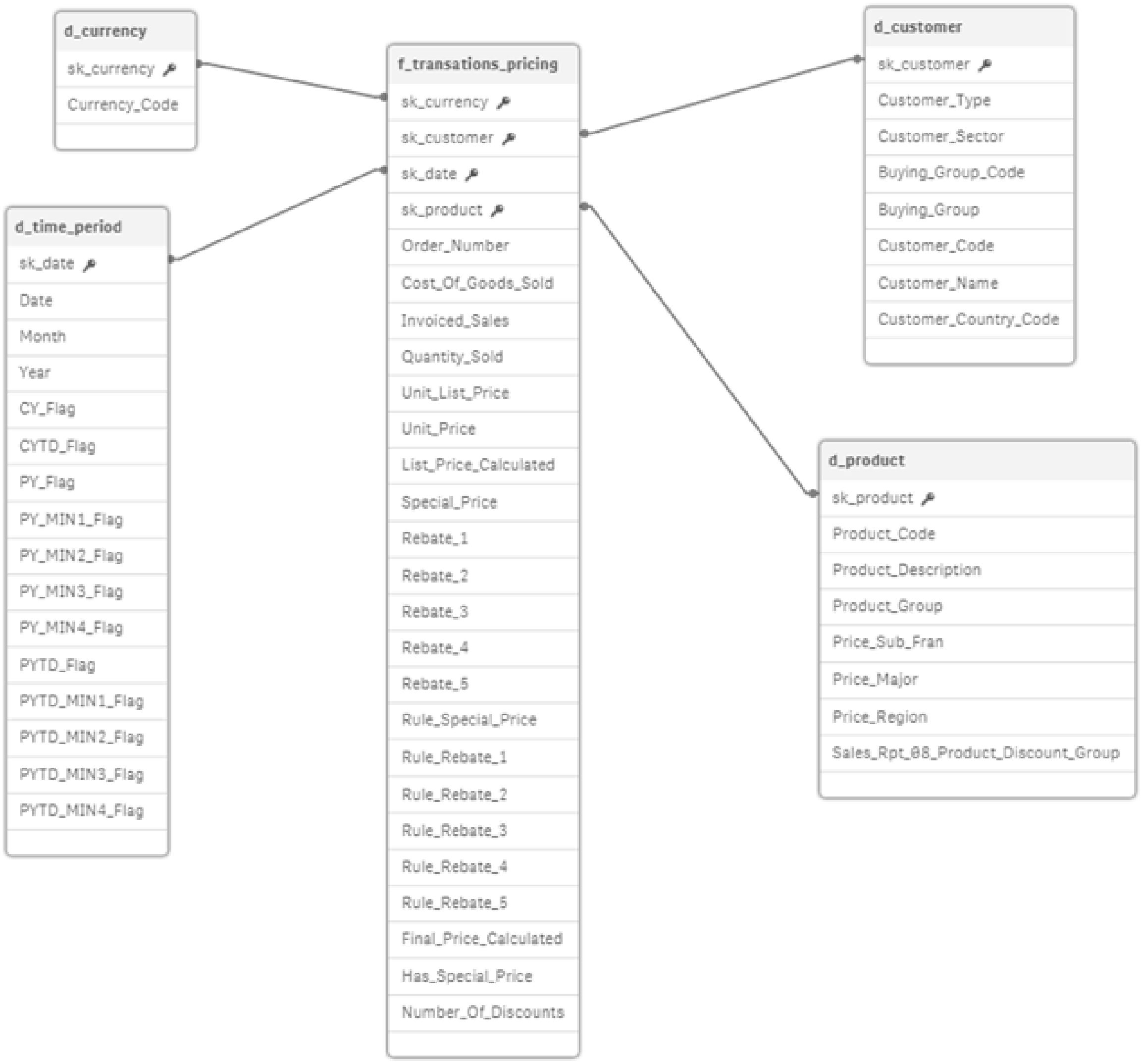


Figure 168 - Data model with d_time_period