Valter Prykäri

Designing a Data Warehouse for Market Basket Analysis in Retailing

Master's Thesis in Information Systems
Supervisor: Dr. Markku Heikkilä
Faculty of Social Sciences, Business and Economics
Åbo Akademi University
Åbo 2019

TABLE OF CONTENTS

ABSTRACT

Subject: Information Systems
Writer: Valter Prykäri
Title: Designing a Data Warehouse for Market Basket Analysis in Retailing
Supervisor: Dr. Markku Heikkilä

Abstract:

Companies are constantly generating large amounts of data from different business processes. This data is usually spread out in many separate source systems. Organizations can benefit from utilizing this data by performing different types of analytics. However, the data is usually in different formats and it is not viable to build analytical solutions right on top of the source systems. A data warehouse is one solution to this problem. A data warehouse works as a central repository for company data, and it works as a great foundation for all kinds of reporting and analytics.

Retailers can gain insight into purchase patterns by performing market basket analysis on transactional data. With the help of market basket analysis, retailers can find out which products are frequently bought together. This information can be used for various purposes, such as store design, marketing campaigns and recommendations. There are various algorithms available, and it can be challenging to decide which ones to use. Building an automated solution that can perform market basket analysis can be beneficial for a retail business.

This thesis introduces the concept of data warehousing as well as different architectures and modeling techniques. In addition, some market basket analysis algorithms are introduced. To answer the research questions, a data warehouse and a market basket analysis system was designed with the help of scientific literature and personal project experience. The designed system was then implemented to evaluate its functionality and usability.

After developing and testing the system, it was concluded that the proposed architecture works. The system could be further developed to support more data and different types of analytics.

Keywords: Data warehousing, Market Basket Analysis, Affinity Analysis, Dimensional Modeling

Date: 24.4.2019	**Number of pages:** 82

LIST OF FIGURES AND TABLES

ABBREVIATIONS

3NF	Third Normal Form
BI	Business Intelligence
CIF	Corporate Information Factory
DW	Data Warehouse
EDW	Enterprise Data Warehouse
ER	Entity-Relationship
ETL	Extract, Transform, Load
MBA	Market Basket Analysis
MVP	Minimum Viable Product
OLAP	Online Analytical Processing
OLTP	Online Transaction Processing
POS	Point of Sale
RDBMS	Relational Database Management System
SSMS	SQL Server Management Studio
SSIS	SQL Server Integration Services
SQL	Structured Query Language
TID	Transaction ID
T-SQL	Transact-SQL

1 INTRODUCTION

1.1 BACKGROUND

The amount of data generated is constantly increasing and companies are trying to find new ways of utilizing the vast amounts of data they have stored in various source systems. Data is a valuable asset as it can be helpful in decision making. Most retailers have all of their transactional data saved in a transactional database, and applying data mining methods on the data can give valuable insight into customer behavior. This can help to understand customers and improve the marketing strategy.

As transactional data is easily accessible, most companies could benefit from mining this data. However, especially small and medium sized retailers may lack the knowledge and expertise to implement solutions for analytics. Furthermore, it is a challenge to collect the correct data and to prepare it for analysis, as it is usually not feasible to build analytical solutions directly on the source systems. A solution to this problem is to develop a data warehouse. Designing an automated, effective and affordable data warehouse solution and an ETL-pipeline (Extract, Transform, Load) for data mining purposes is therefore crucial.

A common approach to utilize transactional data is to perform market basket analysis. With the help of market basket analysis, it is possible to gain valuable information from the raw transactional data. Market basket analysis can help to identify items that are frequently bought together, and this can be used for various purposes such as store layout design and campaigns. By developing a data warehouse that serves analytical needs it is possible to build an automated system for transactional data analytics.

By personal experience, there is still a lack of knowledge in data warehouse design in the industry, and companies are not aware of the possibilities and opportunities that analytics can provide. As it is difficult to understand the benefits and drawbacks of different data warehouse modeling techniques and market basket analysis algorithms, it is interesting to dive deeper into this area. As market basket analysis can be highly useful for retailers, it is important to research the best way of developing a system for this purpose.

1.2 AIM AND RESEARCH QUESTIONS

The aim of this thesis is to study different data warehousing architectures as well as different data mining methods that are applicable to transactional data to gain insight about customer purchase patterns, namely in the form of market basket analysis. There are various methods available, and it can be quite challenging to know which methods should be used in different scenarios. The aim is to then design a generic data warehouse that can be used as a starting point when designing a data warehouse solution for market basket analysis in the retail industry. The resulting model is not meant to be a complete data warehouse model for a retail company, but rather a part of a solution that can quickly deliver results. The architecture should be easily scalable and support various future BI/Analysis needs. The designed system will also include a market basket analysis solution. The goal is to finally develop the proposed system to test its functionality and analyze the results. I try to answer the following research questions with this thesis:

1. *How should a data warehouse be designed so that it can support market basket analysis?*
2. *How should automated market basket analysis be applied?*

With this thesis, I hope to shed light upon the benefits and drawbacks of the different methods available in literature.

1.3 LIMITATION

My thesis will be limited to data warehousing models and market basket analysis in the retail industry. The thesis will focus on existing theories and models introduced in literature. Challenges in big data will not be part of this thesis. Additionally, I will not try to improve or do performance testing on different algorithms. I will further limit the algorithms discussed in this thesis to Affinity Analysis and Market Basket Analysis. Although data presentation is a crucial part of the models that are studied, the thesis will not focus on data presentation and visualization theory.

1.4 STRUCTURE OF THE THESIS

The second chapter of this thesis presents the research methodology. Chapters three and four are a literature review on data warehousing and market basket analysis algorithms, which in turn serve as a foundation for chapter five of the thesis which consists of the actual design and development of the architecture. Chapter six includes the conclusion of the thesis. Finally, the seventh chapter is a summary of the thesis in Swedish.

2 RESEARCH METHODOLOGY

There are various research methods that are relevant and used in information systems research. Exploratory research is suitable for problems that have not yet been studied. It will help in defining and understanding a problem. This method will not lead into conclusive results. In this case, the research will start with a general idea, and the results can be used for future research. Empirical research can be used to answer clear questions and it can be both quantitative and qualitative. Quantitative research methods include conducting surveys and gathering data to perform statistical analyses. The data can be used to uncover patterns. Qualitative methods on the other hand have typically smaller sample sizes. Some examples on qualitative methods are interviews, observations and focus groups. A common method is to conduct a case study, which can be both quantitative and qualitative.

Järvinen (2004) have presented a taxonomy for information systems research, illustrated in Figure 1. Innovation-building and innovation-evaluating approaches are related to building and evaluating e.g. an information system, which makes this branch of the taxonomy relevant to the thesis. An innovation building approach is most relevant, as the thesis will focus on answering the research questions without an existing information system. The methodology for the thesis is based on design science research. Design science is a suitable approach, as it is appropriate to research the problem in this thesis by designing an artifact and actually implementing it to test its functionality and evaluate it.

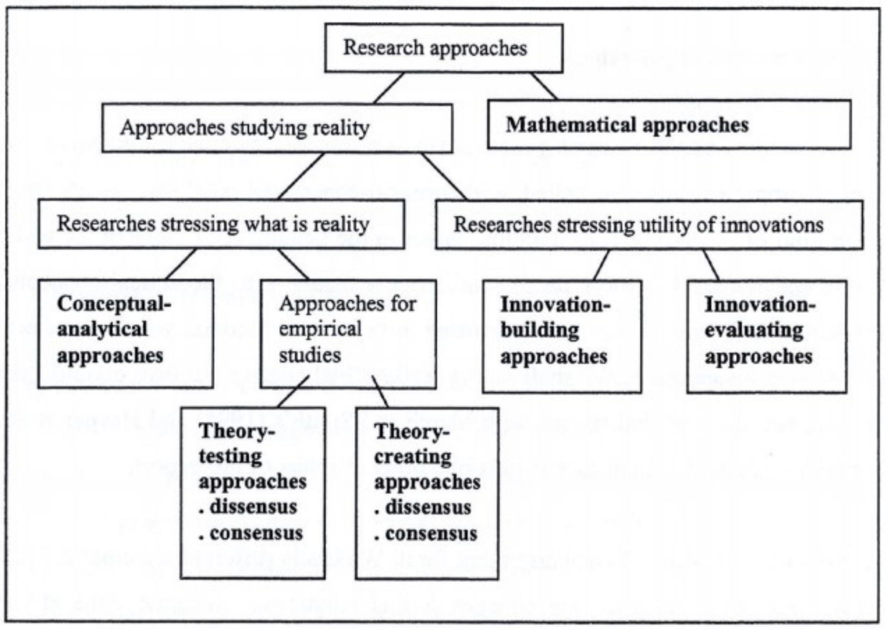

Research approaches
Approaches studying reality
Mathematical approaches
Researches stressing what is reality
Researches stressing utility of innovations
Conceptual-analytical approaches
Approaches for empirical studies
Innovation-building approaches
Innovation-evaluating approaches
Theory-testing approaches
. dissensus
. consensus
Theory-creating approaches
. dissensus
. consensus

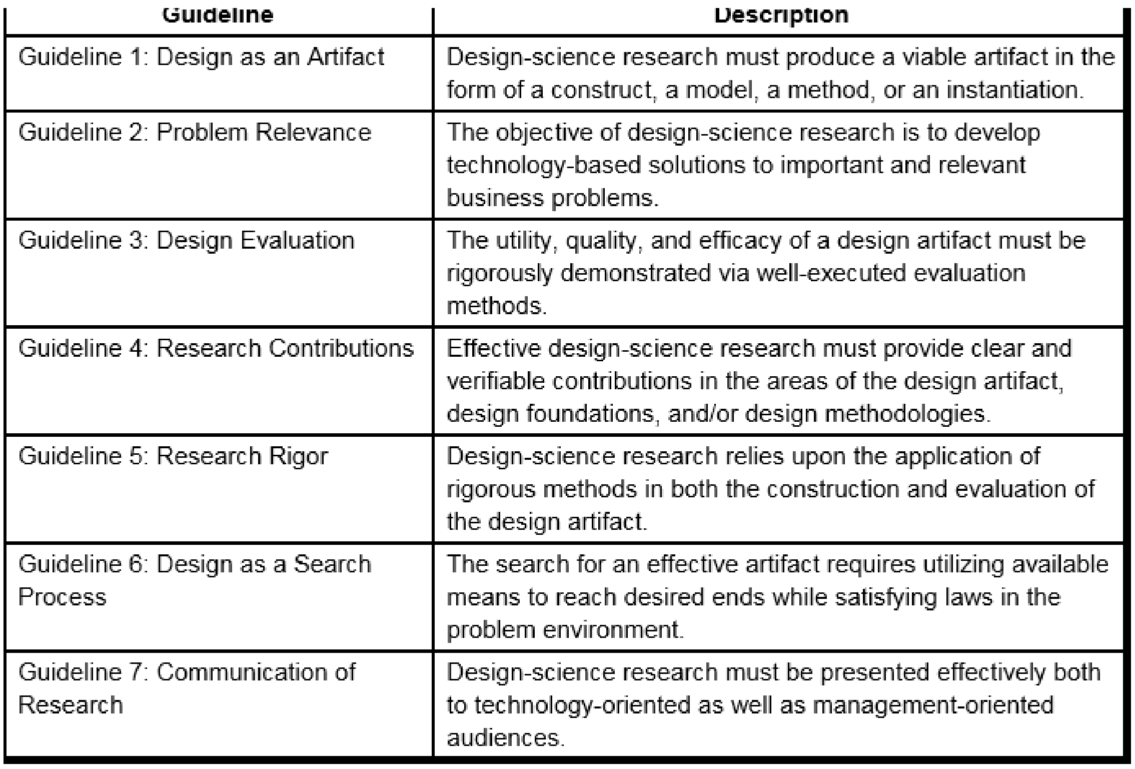
Figure 1. Taxonomy of research methods. (Järvinen, 2004)

2.1 DESIGN SCIENCE RESEARCH

According to Järvinen (2004), design science research is relevant when we try to build or evaluate an innovation. Hevner et al. (2004) have stated that design science research is a common approach in information systems research and "seeks to extend the boundaries of human and organizational capabilities by creating new and innovative artifacts". Design science is a problem solving process. The main goal of design science research is to acquire knowledge and understanding of a design problem by building and applying an artifact. The result of design science research is therefore an artifact that addresses an important organizational issue. The artifact should be described in a way that it is implementable and applicable in a relevant domain. Table 1 describes some general guidelines for design science research.

The designed artifact should be evaluated by well defined evaluation methods and it is a crucial part of the research process. The evaluation is based on the requirements of the business environment, and the environment includes the technical infrastructure meaning that the evaluation of the artifact includes the integration of the artifact into the technical architecture of the business setting (Hevner et al., 2004).

For this thesis, the architecture of the system is the artifact, and it will be evaluated through an actual implementation using test data. The functionality of the system will work as evaluation criteria, as it is not based on a real-world case. The artifact is more of a proof-of-concept that can be implemented in a real business setting.



Guideline	Description
Guideline 1: Design as an Artifact	Design-science research must produce a viable artifact in the form of a construct, a model, a method, or an instantiation.
Guideline 2: Problem Relevance	The objective of design-science research is to develop technology-based solutions to important and relevant business problems.
Guideline 3: Design Evaluation	The utility, quality, and efficacy of a design artifact must be rigorously demonstrated via well-executed evaluation methods.
Guideline 4: Research Contributions	Effective design-science research must provide clear and verifiable contributions in the areas of the design artifact, design foundations, and/or design methodologies.
Guideline 5: Research Rigor	Design-science research relies upon the application of rigorous methods in both the construction and evaluation of the design artifact.
Guideline 6: Design as a Search Process	The search for an effective artifact requires utilizing available means to reach desired ends while satisfying laws in the problem environment.
Guideline 7: Communication of Research	Design-science research must be presented effectively both to technology-oriented as well as management-oriented audiences.

Table 1. Guidelines for design science research. (Hevner et al., 2004)

3 DATA WAREHOUSING FOR RETAIL DATA

This chapter will give an introduction to the concept of data warehousing. The idea behind data warehousing is explained, different models and architectures are introduced and finally benefits and drawbacks of the models are discussed.

3.1 DEFINITION

The concept of Data Warehousing (DW) is not new as it has already been around for three decades. The need for Data Warehousing originally appeared as companies had a need for information. As the multiple systems in companies started to look like giant spider webs, there had to be a better solution for making good corporate decisions than trying to navigate the data in all the various sources (Inmon et al., 2008).

Inmon et al. (2008) define the Data Warehouse as “a basis for informational processing”. Further, a Data Warehouse is subject-oriented, integrated, non-volatile, time-variant and a collection of data in support of management's decisions. The most important task of a data warehouse is to gather and maintain historical data from various sources. This data can consist of many different types and is often generated by various business events and activities (Bojičić et al. 2016). As the data is integrated, it allows for an organization-wide view of the data and an analyst can look at the data as it came from one single source (Inmon et al., 2008). An Enterprise Data Warehouse (EDW) is an organization-wide data warehouse that tries to represent all the data and business rules in the organization (Linstedt & Olschimke, 2016)

According to Kimball & Ross (2013) there are a few key goals that a Data Warehouse/Business Intelligence system should solve and they can be extracted from the following themes that have existed for more than three decades:
- Companies collect data, but cannot access it.
- Users need to slice and dice the data in various ways
- Business users need to access the data easily
- Companies want to see the important information only
- Companies spend time battling inconsistencies in the data instead of making decisions

- Companies want people to do more fact-based decision making.

These problems give a good view of the challenges the companies face when trying to make the best decisions for the business. As a result, these concerns still work as the main requirements for a DW/BI system.

According to Kimball et al. (2013), the concerns above can be turned into the following DW/BI system requirements.

- Information has to be easily accessible. The contents of the system should be obvious and easy to understand for the business user. The data labels and structure of the system should take the business users thought process into account and use familiar vocabulary. Business users will also want to utilize the data in various ways, and the results should be accessible with short query times. All this can be summarized into the following statement: Simple and Fast.
- Information must be presented consistently. All data in the system should be credible and carefully cleansed, of high quality and released only when it is ready for the use by the business users. Data labels must also be consistent; same labels must mean the same thing.
- The system must be able to handle changes. Business requirements, user needs, data and technologies will change under the lifespan of the system. This means that the system must be able to adapt to new requirements without invalidating old data. Adding new data or features to the system should not break existing solutions or change the data. In case that descriptive data has to be modified, these changes should be transparent to the users.
- Data must be presented in a timely way. As the data in a Data Warehouse is used for operational decisions, raw data has to be converted into valuable insight in the matter of hours, minutes or seconds. This means that the business and developers must have realistic expectations.
- The information in the Data Warehouse must be safe. Many businesses store sensitive information about customers, price etc. This information must be securely stored in the DW and access to it should be controlled effectively.
- The system should work as an authoritative and trustworthy foundation for

decision making. The data warehouse should have the right data needed for decision making. As the most important outputs of a Data Warehouse is decisions, designing a data warehouse should mean that you are designing a decision support system.

- The business community must accept the Data Warehouse. If the business users do not use the system, it is unsuccessful. It does not matter how nice the solution is if the business cannot use the DW for its intended purpose. DW systems are often optional to use, so a bad solution will not get any or little use.

These requirements can be met by focusing on the following things when designing the Data Warehouse:

- Understand the business users
- Deliver high quality, relevant and accessible data to the business users
- Maintain the Data Warehouse

A typical architecture of a data warehouse is two-layered and introduced by Kimball. This type of architecture is illustrated in Figure 2. The complete data warehouse itself consists of a staging layer and the data warehouse layer. The data from the source systems is first loaded into to staging layer. The staging layer is modeled based on the sources, as the goal is to have an exact copy of the data that is loaded into the data warehouse. The staging area is used to reduce the operations and load times on the source systems (Linstedt & Olschimke, 2016). The transforms and calculations are performed when the data is loaded from the staging layer into the data warehouse. This way the work is focused in the data warehouse itself instead of burdening the source systems.

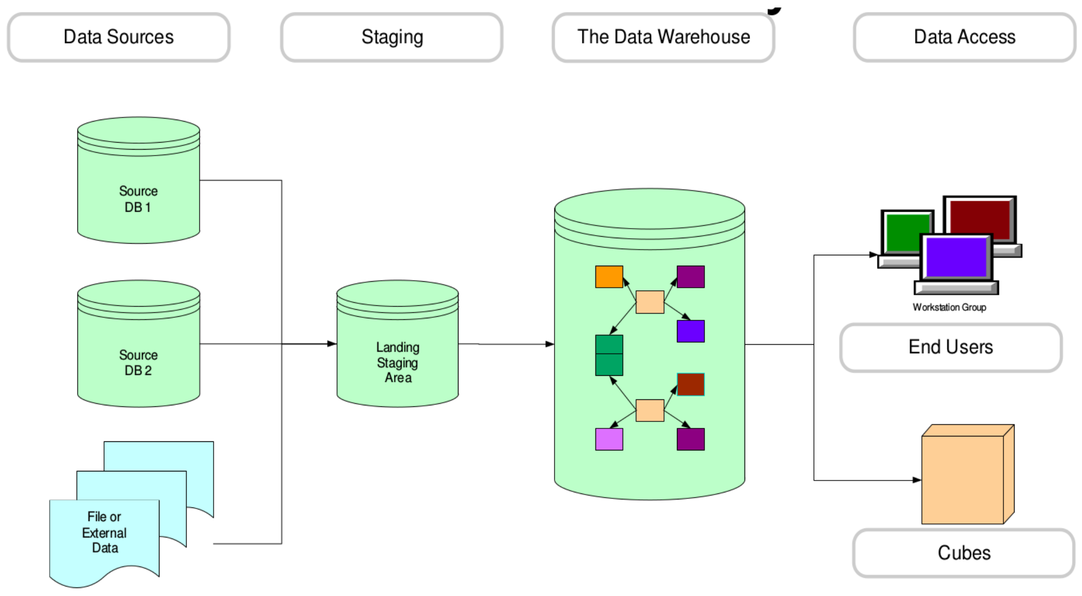


Figure 2. Two-layer architecture. (Abramson)

Another option is a three-layer architecture illustrated in Figure 3. This architecture has been introduced by Inmon. In this architecture, the middle layer holds the atomic raw data that is modeled in 3NF. The goal of this layer is to capture all data in the organization, and it is based on the sources. This layer reminds more of a large operational database. On top of this normalized layer, there is a data mart layer. This data mart layer is most often based on dimensional modeling (Linstedt & Olschimke, 2016).

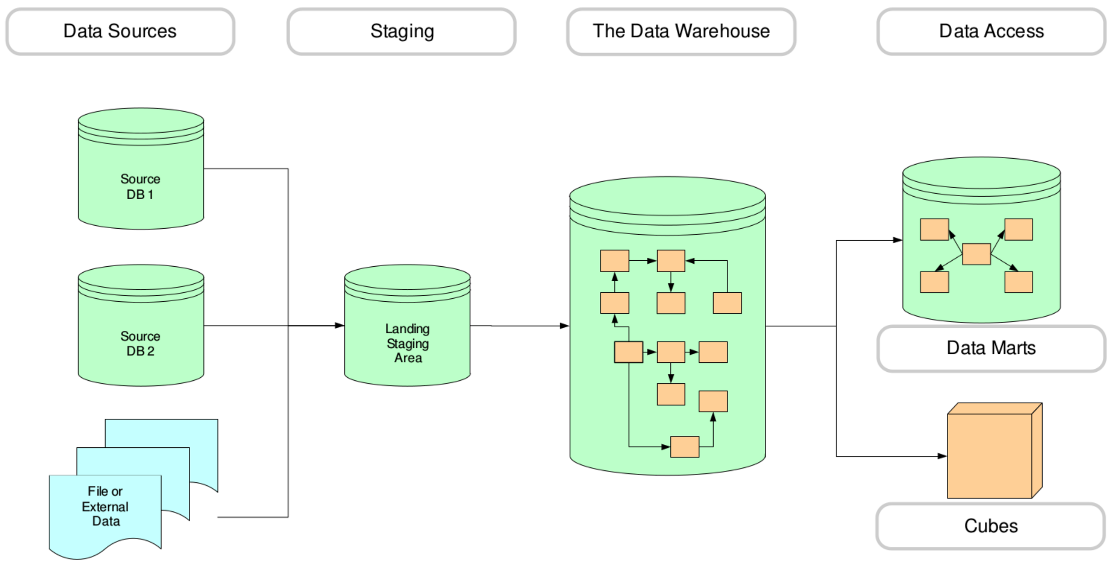


Figure 3. Three-layer architecture. (Abramson)

These architectures and approaches are discussed further in the following chapters.

According to Linstedt & Olschimke (2016), the best way for data warehouse development is an iterative approach. This means that the data warehouse solution is not developed in a “big-bang”-fashion. A “big-bang” approach means that the system is developed as a whole first before the entire system is deployed and ready for use. Instead, the data warehouse should be developed in small iterations. There are still problems in an iterative approach, as the effort to add new functionalities keeps increasing due to dependencies to existing features. Figure 4 illustrates the increased effort when developing subsequent data marts. Every time a new data mart or functionality is developed, it falls into maintenance mode and the development team needs to make sure that all dependencies are taken care of and the old data marts are tested as well. New sources can also cause problems, and it is quite common that the entire solution needs to be refactored when new data marts or functionality is added to the data warehouse solution.

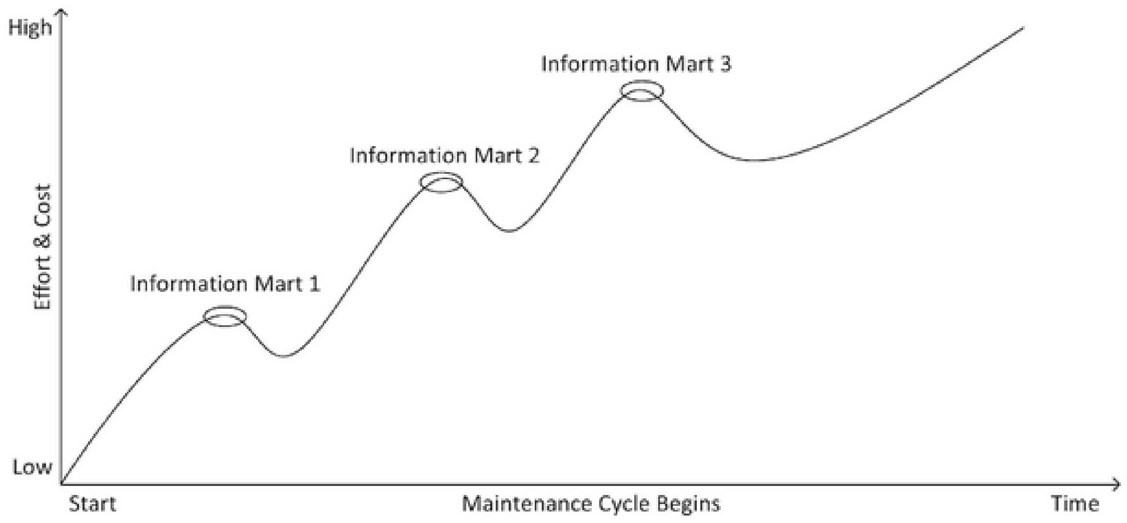


Figure 4. Effort and cost with subsequent data marts. (Linstedt & Olschimke, 2016)

3.1.2 EXTRACT, TRANSFORM, LOAD

As the data needed for any type of analytics is usually in many different sources and in different formats, the source data cannot be used as-is. This is where the Extract, Transform, and Load process comes into play. According to Kimball & Ross (2013), everything between the DW presentation area and the source systems is considered the ETL-system. Extraction is the process of getting the source data from various sources into the ETL system for further processing. The source data is loaded into a staging layer.

After the data has been extracted, the data has to be transformed in numerous ways. Some examples include: correcting spelling errors, standardizing values and formats, and handling incorrect and missing information and adding default values (Inmon et al., 2008, Kimball & Ross, 2013).

The load step refers to the loading of the transformed data into the presentation area in the final layer of the data warehouse. Behind the ETL process there is usually table denormalization, code lookups and splitting or combining columns (Kimball & Ross, 2013).

3.2 DIMENSIONAL MODELING

According to Kimball & Ross (2013), dimensional modeling is widely accepted as the preferred technique for data warehouse design. Linstedt & Olschimke (2016) also states that dimensional modeling is "de-facto standard". This technique addresses two requirements:

- Deliver data that's understandable to the business user
- Deliver fast query performance

Dimensional modeling has been used for a long time because it is a working technique for making databases simple. Simplicity is a basic need of business users, as this helps in understanding and navigating the data that is available. When a data model has a simple design at the start, it has a high chance of staying simple when development continues. A complex design at start will also be complex in the end, and this will cause poor understanding of the model and slow query performance (Kimball & Ross 2013).

Dimensional models are quite different from normal transactional databases which are in 3NF. Databases in 3NF divide data into multiple relational tables which seek to reduce redundancy. This results in a web of relational tables up to hundreds of tables which is highly effective for a transactional system, but hard to navigate as a user. 3NF models work well as operational databases, as each event or transaction usually only does an update or insert in one place of the database. However, it is challenging to build Business Intelligence solutions or queries on this type of database as they are too complex. Users can not understand, navigate or remember the relationships and meanings in tables, and the resulting queries can lead to extremely poor query performance. On the other hand, a dimensional model contains the same information as a normalized model, but the data is formatted in a way that is much easier to understand and provides better query performance (Kimball & Ross, 2013).

Dimensional Models that are implemented in a relational database management system are called star schemas. This is because the model has a star-like structure. This table structure is illustrated in Figure 5 below.

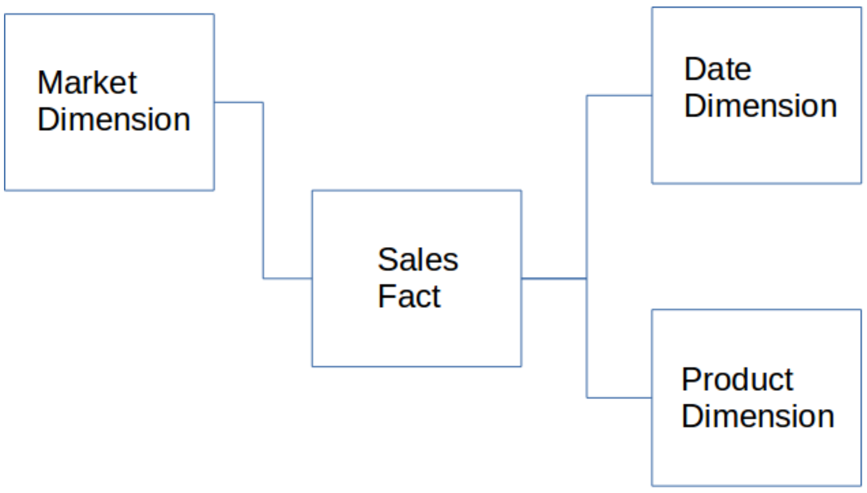


Figure 5. Star Schema.

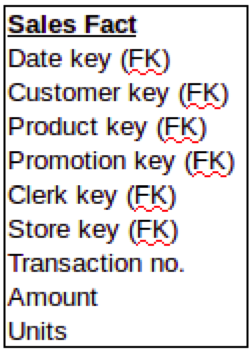
The fact table is the core in the dimensional model and it stores the performance measurements resulting from business process events. Low-level measurements from a single business process should be stored in one dimensional model as measurement data is the largest set of data. This is why the same measurements should not be replicated into multiple places in the data warehouse. By keeping the same data in a single place, it is easier to find the correct data and to maintain consistency throughout the organization. The term *fact* represents measure, and each measure is represented by a row in the fact table. Each row is at a specific level of detail, such as one row per product sold in a transaction. All rows in a fact table must be on the same level of detail, also called the grain. Having rows on different grains can cause problems like double counting (Kimball

& Ross, 2013).

The best facts are numeric, as they can aggregated in multiple ways. Some aggregation examples include adding, calculating averages and finding minimum/maximum values. A good example of a numeric fact is amount in Euros. Additivity is important as analytics and BI-tools usually do not use rows alone, instead they are analysed as larger datasets. However, there are many situations in which facts are also semi-additive and non-additive. It is possible for a fact to be in text form, but in these cases the designer should try to put the text values into dimensions as the values are often from a discrete list of values. If the text is unique for each row in the fact table, it should be added to the dimension table instead. True text facts are problematic to analyze, as aggregation functions are not possible for this type of data (Kimball & Ross, 2013).

Figure 6 shows an example of a simple fact table in a retail sales environment. If there is no sales activity, no new rows will be added to the fact table. This means that no 0-valued rows should be added to the fact table when there has been no activity to show this non occurring activity. By leaving empty rows out of the fact table, it stays scarcer. This is important, as most of the data in a dimensional model is stored in fact tables. Fact tables are in normal cases quite narrow (few columns) but have many rows instead. The grain in a fact table can be of three different types: transaction, periodic snapshot and accumulating snapshot (Kimball & Ross, 2013).

A transaction fact table is a measurement taken at a single activity. One example of this is a retail store beep. The measurement is valid only for that instant, and there is no guarantee for when the next measurement will happen. This type of fact tables are the most dimensional and expressive fact tables (Kimball & Ross, 2013).



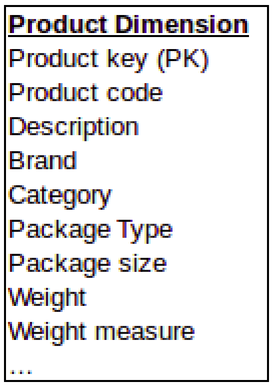
Sales Fact
Date key (FK)
Customer key (FK)
Product key (FK)
Promotion key (FK)
Clerk key (FK)
Store key (FK)
Transaction no.
Amount
Units

Figure 6. Sales fact table. (Kimball & Ross, 2013)

Periodic snapshot fact tables summarizes many measurements over a standard period. As this can be e.g. once a day, week or a month, the grain is the period instead of a single transaction. If no activity takes place in the period, a fact row is typically still inserted with a null or zero value. An accumulating snapshot fact table is quite different from the previous two types; it is used for processes that have a defined beginning and end, and has identifiable milestones in between. The fact row will contain date foreign keys for each milestone, and the fact row is updated correspondingly whenever the row has moved forward in the process (Kimball & Ross, 2013).

All fact tables have foreign keys (marked "FK" in Figure 6) that link to the dimension tables' primary keys. In the sales example, the customer key always matches to a customer in the Customer dimension. When all the foreign keys match to primary keys in their respective dimension tables, database integrity is maintained. The dimensions are accessed through the fact table using joins (Kimball & Ross, 2013).

The dimension tables are crucial for the dimensional model. The dimension tables contain all the additional textual information for the fact table. The dimension table answers the questions "who, what, where, when, how and why" to the event inserted into the fact table. Figure 7 illustrates an example of a product dimension table. Dimension tables are often quite wide with many columns, and opposing the fact table, they have much less rows. Dimension tables can however also contain only a handful of columns as this completely depends on the business and dimensional model (Kimball & Ross, 2013).



Product Dimension
Product key (PK)
Product code
Description
Brand
Category
Package Type
Package size
Weight
Weight measure
...

Figure 7. Product dimension table. (Kimball & Ross, 2013)

Each dimension table has a defined primary key. The purpose of the primary key ("PK" in Figure 7) is to maintain integrity in the data warehouse, as it is joined to the corresponding key in the fact table. The attributes in the dimension tables serve as constraints, groupings and report labels. When a business user wants to see sales per product category, they can simply join the fact table to the product dimension and group by the category attribute available in the dimension table. As the dimension attributes provide this kind of information, they are critical to an understandable and functional data warehouse that is based on a dimensional model. The attributes should consist of logical words that are easy to understand. This also means that any type of codes or abbreviations should be avoided, and full descriptions should be used instead. One should focus on giving verbose and accurate values for the attributes as high quality attributes deliver better analytical capabilities (Kimball & Ross, 2013).

It can be hard to determine whether numeric data is a fact or a dimension. One option to classify them is to make it a fact if it participates in calculations, and make it a dimension attribute if it is part of constraints and row labels. Sometimes, a value seems like a constant attribute but changes so often that it can also be considered a fact. Other times, one might not be sure whether the value is a fact or a dimension attribute and in these cases it can be modeled both ways (Kimball & Ross, 2013).

Dimension tables can often include hierarchical data. In a product dimension, the attributes could include brand name and category name. In these cases, there will be redundant data in the table, but this will in turn lead to easy use and better query performance. It is easy to normalize these situations by creating a separate look-up table. This type of normalization is called snowflaking. As the dimension tables are usually quite small compared to the fact tables, normalization will have low impact on the overall size and performance on the database but will in turn make querying harder. Snowflaking should thus be avoided (Kimball & Ross, 2013).

By finally combining the building blocks, we get a star schema. Figure 8 illustrates a simple star schema for a retail case.

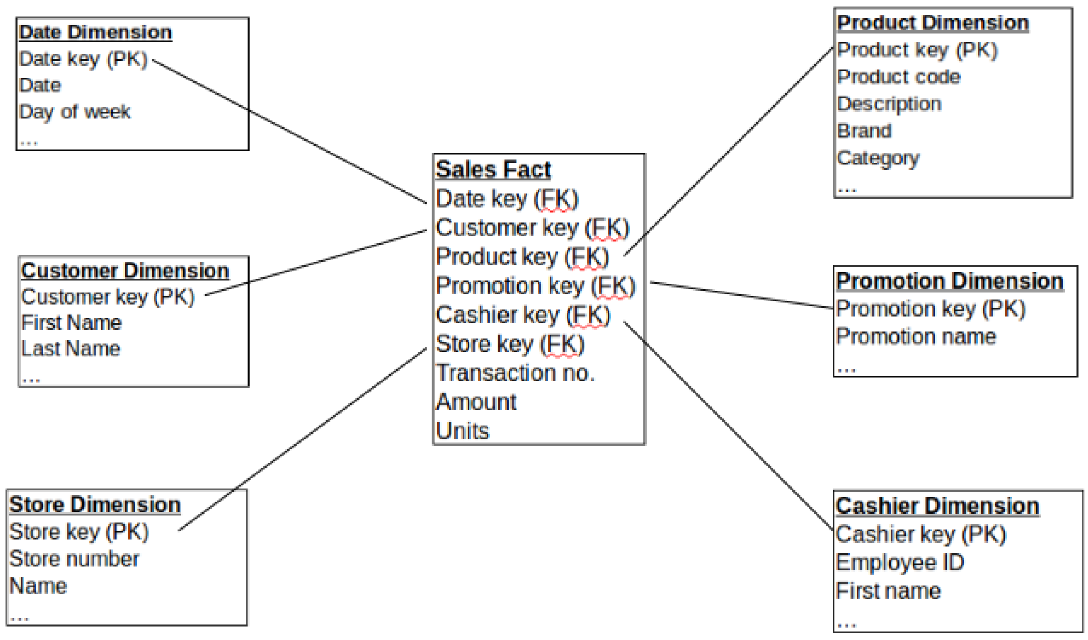


Figure 8. Retail star schema. (Modified from Kimball & Ross, 2013)

Looking at the schema as a whole, it is easy to see and understand its simplicity. Database engines optimize queries on these schemas more efficiently as there are less joins. As the design is simple, it is easy to build reports and analytics on the data. Additionally, the dimensional model can handle changes quite well. As all dimensions work as entry points to the fact tables in the same way, there are no unexpected problems caused by queries (Kimball & Ross, 2013).

The most atomic data is the best for a dimensional model. When the data has not been aggregated, it supports all kinds of unexpected ad-hoc queries and analyses. The dimensional model also supports addition of dimensions and columns without breaking any solutions and integrations that are built on top of the dimensional model (Kimball & Ross, 2013).

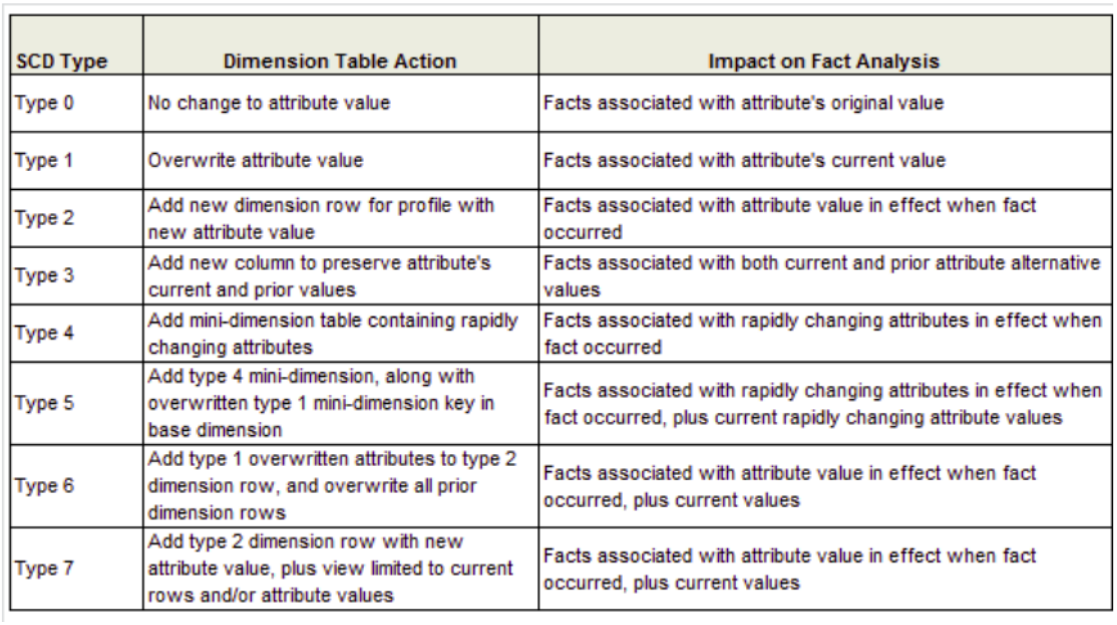
There are several different types of dimensions. The dimensions can either be rapidly changing, meaning that one or more rows in the dimensions change frequently, or a slowly changing dimension. Kimball has defined 7 different types of slowly changing dimensions:

- Type 0, Retain original: In this type, the original attributes will be retained and never changed in the dimension.
- Type 1, Overwrite: The old attribute value will be overwritten by new values, meaning that the dimension will always contain the most recent information.
- Type 2, Add new row: A new row with the new attribute values is added to the dimension. A new primary key should be generalized beyond the natural key as there might be many rows describing the same thing. Type 2 dimensions require three additional attributes: row effective time, row expiration time and current row identifier.
- Type 3, Add new attribute: A new attribute is added to the row that overwrites the old value with the main value. This way the original value can also be stored. This type of dimension is rarely used (Kimball & Ross, 2013; 1Keydata), and should only be used when the attribute values can change a finite amount of times (1Keydata)
- Type 4, Add mini-dimension: This type is used when the attributes change rapidly in the dimension. The dimension is split off to a mini-dimension. The mini-dimension gets an own primary key, and both the main dimension and mini-dimension are accessed through the foreign keys in the fact table (Kimball & Ross, 2013; disoln.org).
- Type 5, Add mini-dimension and outrigger dimension type 1: This type builds on Type 4 for by using a Type 1 outrigger dimension instead of linking through the fact table. The outrigger dimension means that the mini-dimension is accessed through the base-dimension.
- Type 6, Add Type 1 Attributes to Type 2 Dimension: This type builds on Type 2 by adding the current values as attributes to the dimension. These attributes are

systematically overwritten when the value is updated.

- Type 7, Dual Type 1 and Type 2 Dimensions: This hybrid approach allows accessing both historical and current values by having 2 separate dimensions which are accessed by separate foreign keys in the fact table.

The choice of dimension types depend heavily on the data and the needs of the organization. Table 2 illustrates the different types and their impact on fact analysis.

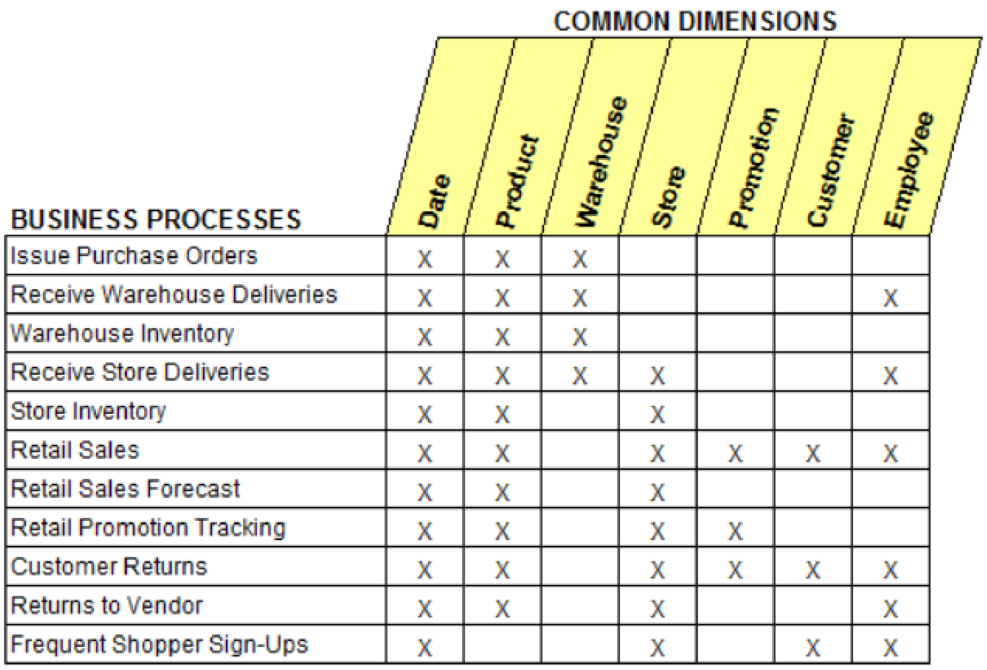


SCD Type	Dimension Table Action	Impact on Fact Analysis
Type 0	No change to attribute value	Facts associated with attribute's original value
Type 1	Overwrite attribute value	Facts associated with attribute's current value
Type 2	Add new dimension row for profile with new attribute value	Facts associated with attribute value in effect when fact occurred
Type 3	Add new column to preserve attribute's current and prior values	Facts associated with both current and prior attribute alternative values
Type 4	Add mini-dimension table containing rapidly changing attributes	Facts associated with rapidly changing attributes in effect when fact occurred
Type 5	Add type 4 mini-dimension, along with overwritten type 1 mini-dimension key in base dimension	Facts associated with rapidly changing attributes in effect when fact occurred, plus current rapidly changing attribute values
Type 6	Add type 1 overwritten attributes to type 2 dimension row, and overwrite all prior dimension rows	Facts associated with attribute value in effect when fact occurred, plus current values
Type 7	Add type 2 dimension row with new attribute value, plus view limited to current rows and/or attribute values	Facts associated with attribute value in effect when fact occurred, plus current values

Table 2. Dimension types. (Kimball & Ross, 2013)

The primary keys in the dimensions should consist of surrogate keys instead of natural or business keys. A surrogate key is an artificial or synthetic key that is used as a substitute for a natural key. The surrogate key should be an integer value which is not a combination of natural keys and it should not be a smart key. A smart key means that the key tells something about the row itself. By using surrogate keys, various errors and problems in production can be avoided as the dimension is not dependent on the keys in the underlying operational systems (Kimball, 1998). These dimension surrogate keys are simple integers assigned in sequence, starting with the value 1, every time a new key is needed (Kimball Group).

To keep the data warehouse intact and controlled, the Kimball approach requires the so called Bus Architecture. The bus architecture focuses on business processes while delivering integration via standardized conformed dimensions. An essential tool for managing the development of the Enterprise Data Warehouse Bus Architecture is the Bus Matrix, illustrated in Figure 9. The rows of the matrix illustrate business processes while the columns are dimensions. The matrix can also be used to prioritize development according to business needs, as the data warehouse should be developed one row at a time (Kimball & Ross, 2013).



	COMMON DIMENSIONS						
BUSINESS PROCESSES	Date	Product	Warehouse	Store	Promotion	Customer	Employee
Issue Purchase Orders	X	X	X				
Receive Warehouse Deliveries	X	X	X				X
Warehouse Inventory	X	X	X				
Receive Store Deliveries	X	X	X	X			X
Store Inventory	X	X		X			
Retail Sales	X	X		X	X	X	X
Retail Sales Forecast	X	X		X			
Retail Promotion Tracking	X	X		X	X		
Customer Returns	X	X		X	X	X	X
Returns to Vendor	X	X		X			X
Frequent Shopper Sign-Ups	X			X		X	X

Figure 9. Enterprise data warehouse bus matrix. (Kimball & Ross, 2013)

3.3 INMON APPROACH

Bill Inmon is considered one of the original influencers in data warehousing. He has been a proponent of the Corporate Information Factory (CIF) as an alternative to the Kimball approach. According to Inmon et al. (2001), the CIF is generic as it is recognizable across different organizations, but it also has its organization-specific parts that are unique based on the business.

In Inmon's approach, all the organizational data is first stored in a central data warehouse in 3NF. The architecture then incorporates several data marts which serve different business needs and users. These data marts extract their data from the central data warehouse. This architecture ensures that the data stored in the business specific data marts is consistent, as all the data marts have the same source; the atomic data warehouse. This architecture follows traditional database design patterns, as the goal is to have a normalized data warehouse.

The development process follows a top-down approach which means that the data warehouse is built before the data marts. The Inmon approach also uses Entity-Relationship Diagrams in the design. This approach means that the development team needs to be skilled, and that the development is quite complex. As a result, it can take considerable time before there are results to show, but the data will be consistent as a result of this approach (Breslin, 2004). Inmon's idea is that the initial effort to build an atomic data warehouse is worth it because this allows building of new data marts for every business need while also keeping consistency in the data (Inmon et al. 2001).

Inmon's approach targets IT Professionals, and requires a substantial knowledge to use his tools and methodologies. This puts the end users more in the role of audience, as the end-users will rely on the output of IT-professionals (Breslin, 2004).

3.4 DATA VAULT

Data Vault modeling is invented by Dan Lindstedt. He started developing and researching this method in 1990, and continued developing it until the early 2000's. The Data Vault 1.0 has later developed into Data Vault 2.0. Data Vault 1.0 focused primarily on the physical and logical models for creating the raw data warehouse. Data Vault 2.0 has expanded and also includes methodology, architecture and implementation. Methodology refers to agile development methods and the architecture includes NoSQL and big-data systems (Linstedt & Olschimke, 2016).

The Data Vault 2.0 architecture illustrated in Figure 10 implements a modified three-layer architecture as introduced earlier in this thesis. The architecture consists of three layers. The first layer is the staging area, which collects the raw data from source systems. The data warehouse layer consists of a data warehouse modeled according to the data vault

2.0 modeling technique. The final layer in the architecture includes one or more data marts that are used for data information delivery and they are modeled as star schemas or other structures depending on the needs of the data warehouse. The architecture also allows optional vaults that include operational data, metrics and business rules which are integrated in the data warehouse layer. The architecture can also allow capabilities for self-service BI including write-back of information to the Data Vault (Linstedt & Olschimke, 2016). Due to platform independence, the Data Vault 2.0 architecture also supports NoSQL in all layers.

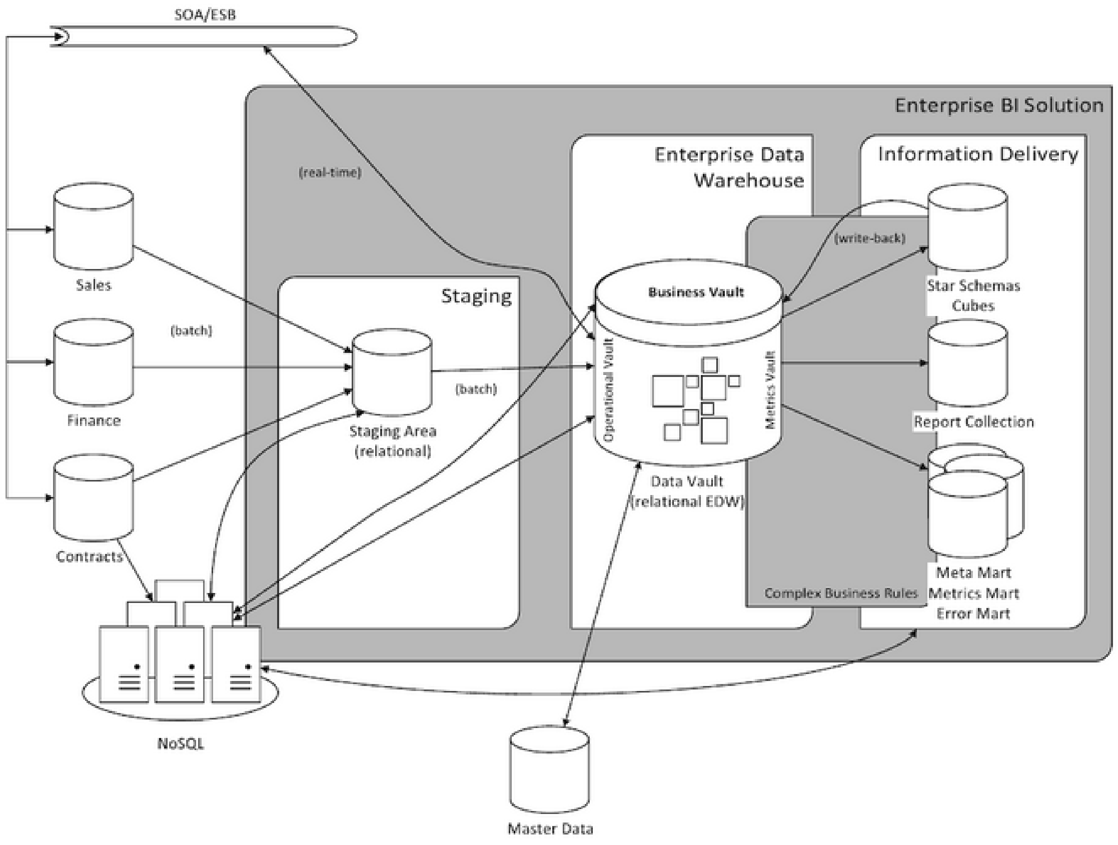


Figure 10. Data Vault architecture. (Linstedt & Olschimke, 2016)

The staging area tables duplicate the source table structures. This means that the tables have the same columns including primary key columns. Indexes and foreign keys are however not included in the staging tables. All columns should be nullable, as also faulty and dirty source data should be loaded into the staging area. In this layer, only hard business rules should be applied. Hard business rules refer to rules like truncating long strings, but are not specific logic for the business. It is also common to include the source table names and columns in the staging area (Linstedt & Olschimke, 2016).

The purpose of the data warehouse layer in the Data Vault 2.0 architecture is to store all historical, time-variant data. The Data Vault stores raw data that has not been modified by any business rules and logic, except for hard business rules. The data is stored at the same grain as in the source. Every change in the source systems is also tracked in the Data Vault (Linstedt & Olschimke, 2016).

As the data warehouse layer in the Data Vault architecture is not directly accessed by the end-users, there is a separate layer with information marts. This way, the data is delivered to the end-users in a more comfortable way. The data in the information mart can be aggregated, flat or wide, structured for reporting, indexed etc. It is often based on dimensional modeling (Linstedt & Olschimke, 2016). Linstedt (2010) has also stated in his blog that the data vault model should not be directly accessed by BI applications. The model is meant to work as a backend in the data warehouse. This means that the data vault is only used to store the historical data in the organization and to provide a platform for quick development of data marts. The data marts can then be built using star schemas.

Data Vault modeling is based on three main entities. These entities are Hubs, Links and Satellites. Figure 11 illustrates a logical design of an airline system.

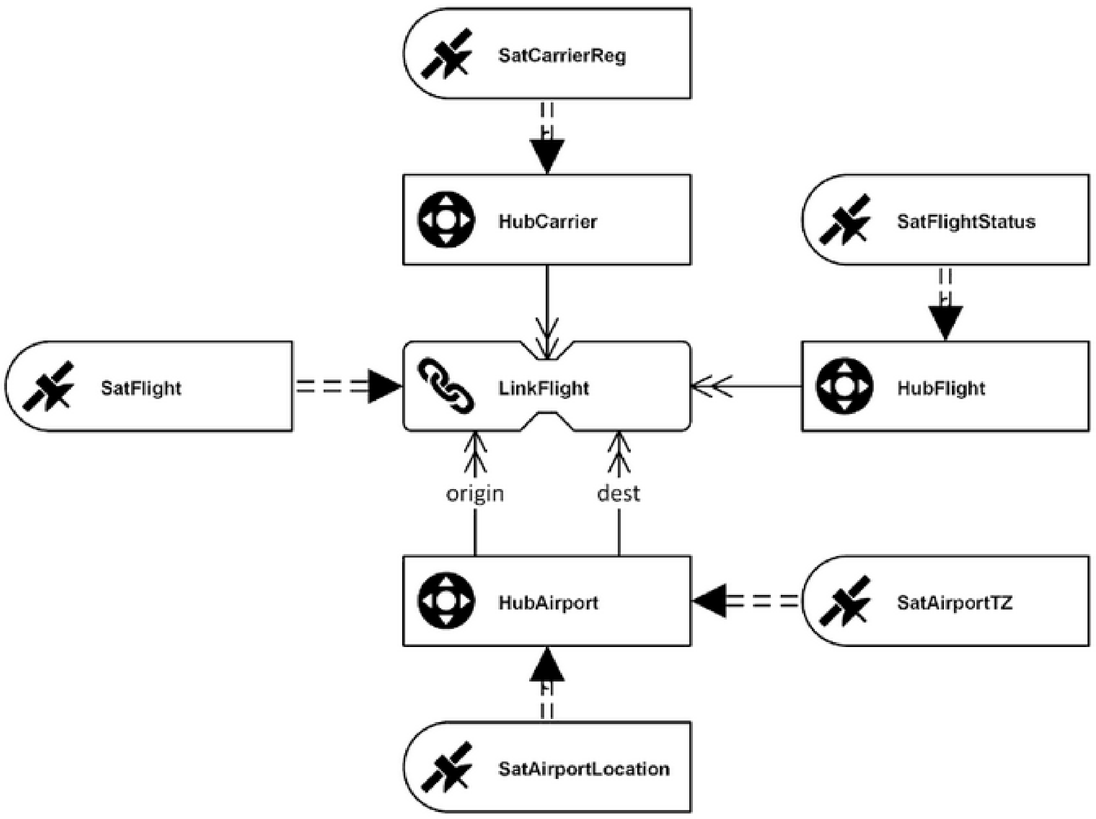


Figure 11. Logical Data Vault design. (Linstedt & Olschimke, 2016)

Every business object has a business key of some sort, be it an invoice number, customer number or an ID. In some cases the business key might be a combination of many fields, or it might be identifiable by a name. As the business keys are a central part of identifying business objects, these are separated in the Data Vault Model. The purpose of the Hub entity is to store these business keys along with some metadata. In the Data Vault model, there are different hubs for each type of business key. Hubs are the foundation of the Data Vault model. Figure 12 illustrates an airline hub. All entries in a hub should have the same semantic meaning and granularity, which means that a contact person should be in a different hub than a customer that is a company. The hub tracks all business keys in the data warehouse with the help of the following additional attributes: source system, load date and a hash key which is used to reference business objects in other entities. (Linstedt & Olschimke, 2016).

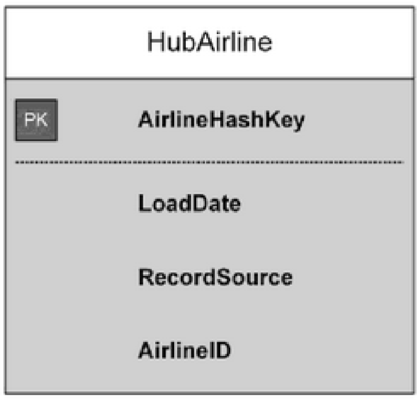


Figure 12. Airline Hub. (Linstedt & Olschimke, 2016)

Business objects are never entirely separate from other business objects, instead they are naturally linked to each other in any business setting. The purpose of the link entity is to track the different links that exist between two or more hubs. A business process often represents a transaction, and this transaction is often a link as well. Figure 13 illustrates and example of a link entity. The main structure of a link includes the hash keys of the referenced hubs. In addition, a link has its own hash key and load date. A link can also reference more than two hubs, as seen earlier in Figure 10. Links store the connection between two hubs at the lowest possible grain, and they will store past, present and future data. The links will not store timelines or temporarity, instead they will represent a relationship that exists currently or existed in the past. Links will also ensure scalability in the warehouse, as it is possible to start with a small Data Vault and then develop it further by adding more links and hubs. Due to the link entities, only many-to-many relationships can exist in the Data Vault Model as links can model 1:m, 1:1, m:m, and m:1 relationships without changing the table definitions. This whole structure allows for easy scalability and faster response time from IT when there are changes in the business requirements (Linstedt & Olschimke, 2016).

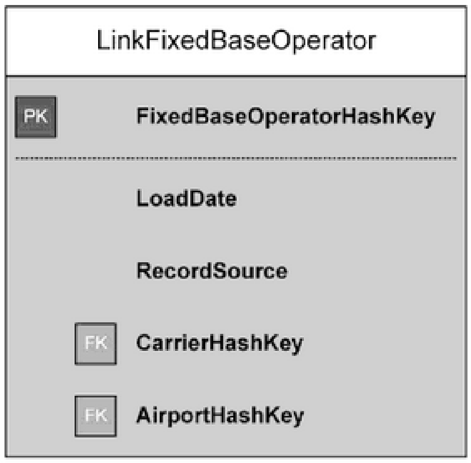


Figure 13. Operator link. (Linstedt & Olschimke, 2016)

With only links and hubs, the Data Vault Model would not provide all information that is needed of a data warehouse. This is where the satellite entities come into play. Figure 14 illustrates a satellite for an airport. The context and attributes of the hubs and the links are stored in satellite entities. There can be multiple satellites on a link or a hub. There are many reasons for distributing the attributes into different satellites, and these include: multiple or changing source systems, different frequency of changes, or functional separation of attribute data. The satellites provide context to links and hubs at a given time, and the descriptive data in a satellite can often change over time. The satellite will also track these changes over time. A satellite is attached to only one hub or a link and is therefore identified by the parents' hash key. In addition to the standard metadata, the satellite entity stores all needed descriptive attributes (Linstedt & Olschimke, 2016).

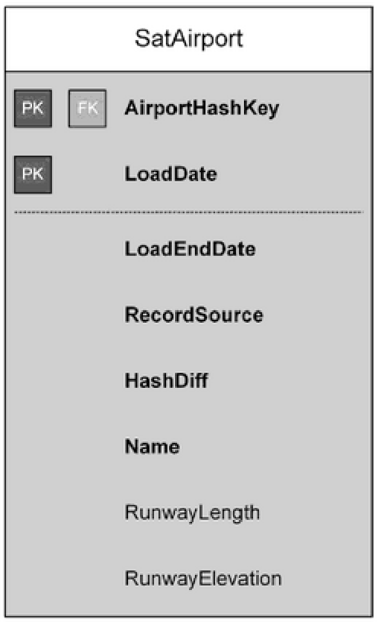


Figure 14. Airport satellite. (Linstedt & Olschimke, 2016)

Due to the architecture of the Data Vault, the only place where historical data is stored is in the Data Warehouse layer. There is no historical data in the staging area, and the data in the data marts may change due to changes in the business requirements. This means that to maintain auditability, no changes should be made to the data in the satellites, with the exception of the load end date attributes.

3.5 CHOICE OF ARCHITECTURE

According to Inmon et al. (2008) there are some fundamental flaws in the dimensional modeling technique. One problem are their brittleness, which means that they are designed for a specific requirement and cannot handle changes well. As soon as there are bigger changes in the requirements, the star schemas require massive changes or even complete remodeling. They are also not easy to extend, as they are limited to the set requirements. Star schemas are also aimed at one audience, meaning that usually the star schemas are optimal for only some of the users, when the data warehouse should be organization-wide. As a single star schema does not satisfy all users, multiple star schemas will cause problems with granularity. To avoid granularity issues, all schemas should be at the lowest grain, but this defeats the purpose as it creates a classical relational design.

The Inmon and Data Vault models will inevitably contain more tables and be more complex than a dimensional model. According to Linstedt (2010) the data vault model is designed for flexibility. As all the relationships are extrapolated to link tables, the design is bound to have more tables than a dimensional model. In return, this design allows for more flexibility, and more joins isn't necessarily a bad thing. When star schemas grow they also have to be conformed, and this can lead to very complex ETL. This will in turn lead to slow loading times and slow development when there are changes in requirements. Linstedt (2010) has also argued that star schemas have scalability issues, especially when the amount of data is in the hundreds of terabytes range.

According to Jukic (2006), the difference between the Inmon and Kimball approaches can be considered a trade-off between extensiveness versus quickness and simplicity. As the Kimball approach only requires dimensional structures without an underlying normalized model, it is inevitably quicker and easier to implement. If the business will only need dimensional structures for their analysis needs, then the Kimball approach is the quicker and easier way to develop the data warehouse. However, the Kimball approach is designed for end-user OLAP-style analysis, so if the organization needs data stores structured in wide variety, the Inmon approach is a better choice. The Kimball approach is sometimes criticized for not being enterprise-wide, but this is simply not true as the fact tables and dimensions can be created enterprise wide. Kimball et al. (2013) have further pointed out that the Inmon approach can by all means be used if the organization has the patience, budget, appetite and need to have a fully normalized structure before loading the data to dimensional structures.

According to Bojičić et al. (2016), the main weakness of the normalized model, or Inmon-approach, is that the relationships and attributes are related to the source system, so any changes in the structure of the source will cause a need for change in the data warehouse structure as well. The Data Vault model, however, offers more flexibility for these types of problems as the structure of an object is decoupled from the object itself with the help of satellites. On the other hand, obtaining the original source model from the data vault model is not possible. The dimensional model suffers from some of the same problems as the normalized model, as addition of new attributes in the source requires changes in the corresponding dimension. Bojičić et al. (2016) concluded that none of the models fulfill all the requirements for a data warehouse. The choice of model should thereby not be based on which methodology is “better”, but rather on the right fit for the project and organization in question (Jukic, 2006).

4 MARKET BASKET ANALYSIS AND AFFINITY ANALYSIS

The goal with affinity analysis is to find attributes that belong together. A popular application of this is through market basket analysis. With market basket analysis, the goal is to seek associations between two or more attributes (Larose 2005). With the help of Market Basket Analysis we can identify frequent patterns in a dataset. Frequent patterns are sets of items, subsequences or substructures that appear frequently in a dataset (Han et al. 2014). With the help of market basket analysis, it is possible to identify which products in a retail store are frequently bought together which can then be used to improve marketing, shop layout, recommendations etc. This chapter will discuss the basics of market basket analysis and some of the methods that are used.

4.1 SALES DATA ANALYSIS WITH ASSOCIATION RULES

A retail store will store various types of data about customers and sales. This data can be analyzed to help the business boost its sales. Finding association rules with the help of market basket analysis can help the company in this task. Association rules are used to find associations and patterns between objects in a dataset. In a retail context, market basket analysis can tell which products are bought together frequently. A retail company can use this information to change or add new products, perform cross-marketing and send customized emails (Jabeen, 2018). Retailers can also use the information to design the store layout to improve the shopping experience and marketing in ways that encourage customers to spend more on their shopping basket (Jabeen, 2018, Karthiyayini & Balasubramanian, 2016). Online retailers and publishers can use affinity analysis for better content placement, drive recommendation engines and deliver targeted marketing by offering products that are likely to be interesting to the customers (Karthiyayini & Balasubramanian, 2016).

4.2 SUPPORT, CONFIDENCE AND LIFT

Let *I* be a collection of itemsets. Suppose that *I* contains two different itemsets, A (e.g milk, bread) and B (e.g. carrots). An association rule is in the following form: if A, then B (A ⇒B), where A and B are mutually exclusive (Larose 2005). The support for a particular association rule A ⇒B is the proportion of transactions in the whole dataset that contain both A and B (Larose 2005, Han et al. 2014). This can be described as follows:

$$support = P(A\ U\ B) = \frac{number\ of\ transactions\ containing\ both\ A\ and\ B}{total\ number\ of\ transactions}$$

The confidence of the association rule A ⇒ B measures the percentage of transactions containing A that also contain B (Larose 2005, Han et al. 2014). This can be written as follows:

$$Confidence\ =\ P(B\mid A) = \frac{P(A\ U\ B)}{P(A)}$$

$$= \frac{number\ of\ transactions\ containing\ both\ A\ and\ B}{Number\ of\ transactions\ containing\ A}$$

To illustrate these concepts, consider Table 3 which consists of a number of transactions.

TID			
1	Bread	Milk	Carrots
2	Bread	Milk	
3	Eggs	Drink	
4	Bread	Milk	Carrots
5	Eggs	Carrots	

Table 3. Example of transactions.

Consider the rule {Bread, Milk} ⇒{Carrots}. As there are 2 transactions containing Bread, Milk and Carrots, the support for the rule is 2/5 = 0,4. As there are 3 transactions containing Bread and Milk the confidence for the rule is 2/3 = 0.67. According to Larose (2005), analysts may prefer rules that have either high confidence or high support depending on the situation and the data that is analysed. Usually rules that have both high support and confidence are preferred. A k-itemset is an itemset containing k items, e.g. {orange,banana} is a 2-itemset. The number of transactions containing a particular itemset is the itemset frequency. When performing analysis, the minimum support and confidence thresholds are set to help determine and identify interesting rules (Jabeen, 2018).

Lift measures the correlation between A and B in the rule A ⇒B. The correlation shows how the itemset A affects the itemset B. Lift is calculated with the following formula:

$$\mathit{Lift}\ (A,B) = \frac{P(A\ U\ B)}{P(A)P(B)}$$

This means that the higher the lift, the higher the chance of A and B occurring together (Jabeen, 2018). A lift of 1 means that A and B are independent, and a lift of < 1 means that the presence of A has a negative effect on B.

4.3 METHODS

In this section I will present common algorithms that are useful and widely used for market basket analysis. First, the apriori-algorithm will be discussed followed by the Eclat-algorithm and FP-growth. Finally, other approaches will be briefly introduced.

4.3.1 APRIORI-ALGORITHM

The Apriori algorithm is a typical algorithm used for frequent itemset mining in transactional data (Yabing 2013, Rathod et al. 2014). The Apriori-algorithm was proposed by Agrawal & Srikant in 1994. The algorithm uses support-based pruning to control the growth of candidate itemsets. The algorithm is shown in Figure 15 (Tan et al. 2006).

```
1: $k = 1$.
2: $F_k = \{ i \mid i \in I \wedge \sigma(\{i\}) \geq N \times minsup\}$.    {Find all frequent 1-itemsets}
3: repeat
4:   $k = k + 1$.
5:   $C_k$ = apriori-gen($F_{k-1}$).    {Generate candidate itemsets}
6:   for each transaction $t \in T$ do
7:     $C_t$ = subset($C_k$, $t$).    {Identify all candidates that belong to $t$}
8:     for each candidate itemset $c \in C_t$ do
9:       $\sigma(c) = \sigma(c) + 1$.    {Increment support count}
10:    end for
11:  end for
12:  $F_k = \{ c \mid c \in C_k \wedge \sigma(c) \geq N \times minsup\}$.    {Extract the frequent $k$-itemsets}
13: until $F_k = \emptyset$
14: Result = $\bigcup F_k$.
```

Figure 15. Apriori algorithm. (Tan et al. 2006)

Initially, the algorithm goes through the whole dataset, and determines the support count for each 1-itemset. After step 1 and 2 are completed, all frequent 1-itemsets are known. Next, the algorithm will generate new candidate itemsets using the frequent (k-1)-itemsets from the earlier iteration using a function called apriori-gen. This function is described in 4.3.1.1. In the steps 6-10, the support count for each candidate itemset is counted by going through the dataset. This function is described in chapter 4.3.1.2 When the support counts for each candidate itemset is known, the algorithm prunes all candidate itemsets that do not meet the minimum support count provided to the algorithm. The algorithm will repeat with k= k+1 until no new frequent itemsets are generated (Tan et al. 2006).

There are some drawbacks in the Apriori algorithm. The candidate generation process takes a lot of time, space and memory, and the algorithm requires scanning of the database multiple times (Kumbhare & Chobe 2014). There are however various ways to improve the performance of the Apriori algorithm. Singh et al. (2013) have introduced an improved algorithm in which the size of the transaction is introduced. This leads to an improvement as the candidate itemsets are reduced as well as I/O by cutting down the amount of transaction records in the database. To improve processing of large data, Li et al. (2012) have implemented a parallel Apriori algorithm based on MapReduce. Their implementation was successful and works well with increasing database size.

4.3.1.1 Apriori-gen

The apriori-gen function consists of two different operations; candidate generation and candidate pruning. In the candidate generation step, new candidate k-itemsets are generated from the k-1 itemsets generated in the previous step. In the candidate pruning step, some of the candidate itemsets are pruned based on the support count they have. The pruning is based on the Apriori principle; if an itemset is frequent, then all of its subsets must also be frequent. To better illustrate this, consider X to be a k-itemset. The algorithm has to determine if the subsets of X are frequent, and if any of the subsets are not frequent, X is immediately pruned. This pruning step reduces the amount of itemsets considered for counting of the minimum support later on in the algorithm. The apriori principle is further illustrated in Figure 16 (Tan et al. 2006). Considering the requirement on frequent itemsets, Figure 17 illustrates the apriori-principle based pruning.

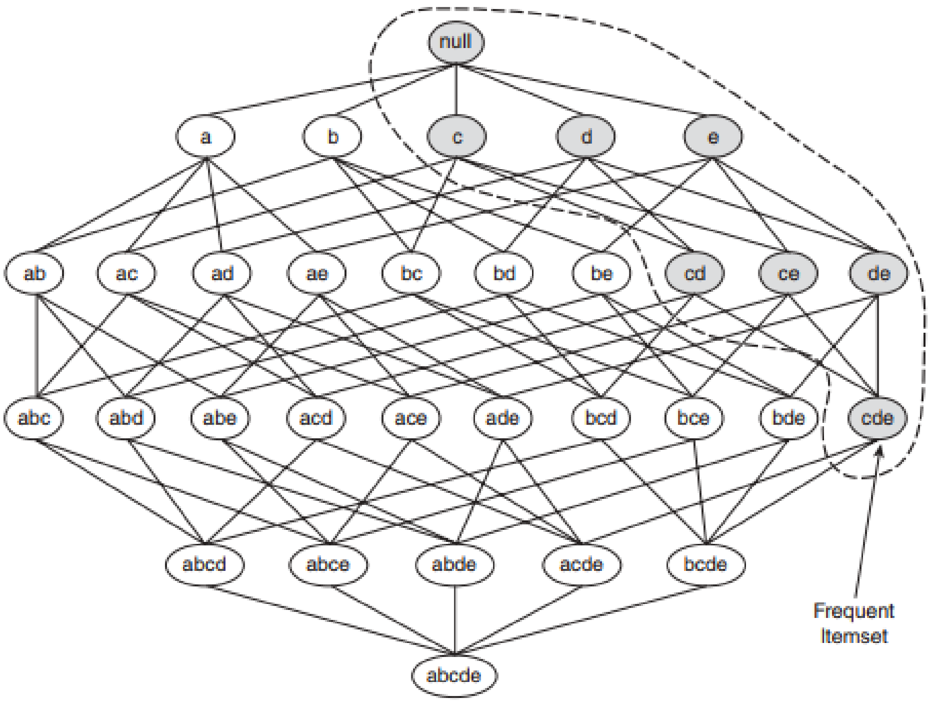


Figure 16. Illustration of the Apriori principle. (Tan et al. 2006)

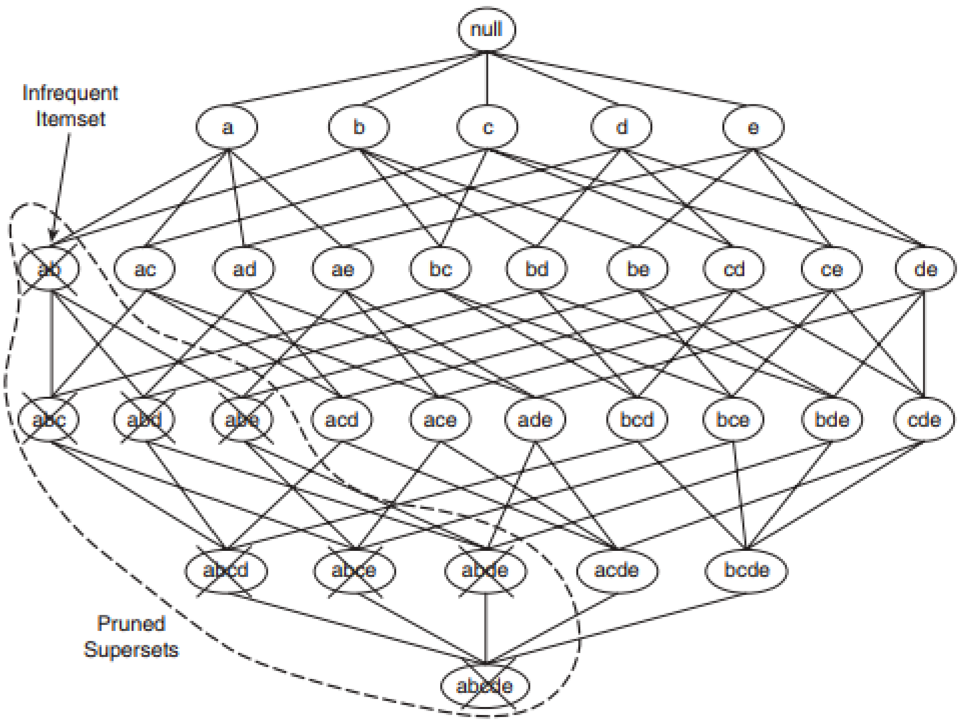


Figure 17. An illustration of Apriori-principle based pruning. (Tan et al. 2006)

There are multiple methods for candidate generation. According to Tan et al. (2006) the apriori-gen function uses the $F_{k-1}\ x\ F_{k-1}$-method, which merges two k-1 itemsets if their k-2 items are identical. The algorithm does not need to merge itemsets that have different first items, as if they are frequent itemsets the same itemset will be generated by merging itemsets with the same first items. This is best illustrated in Figure 18 below.

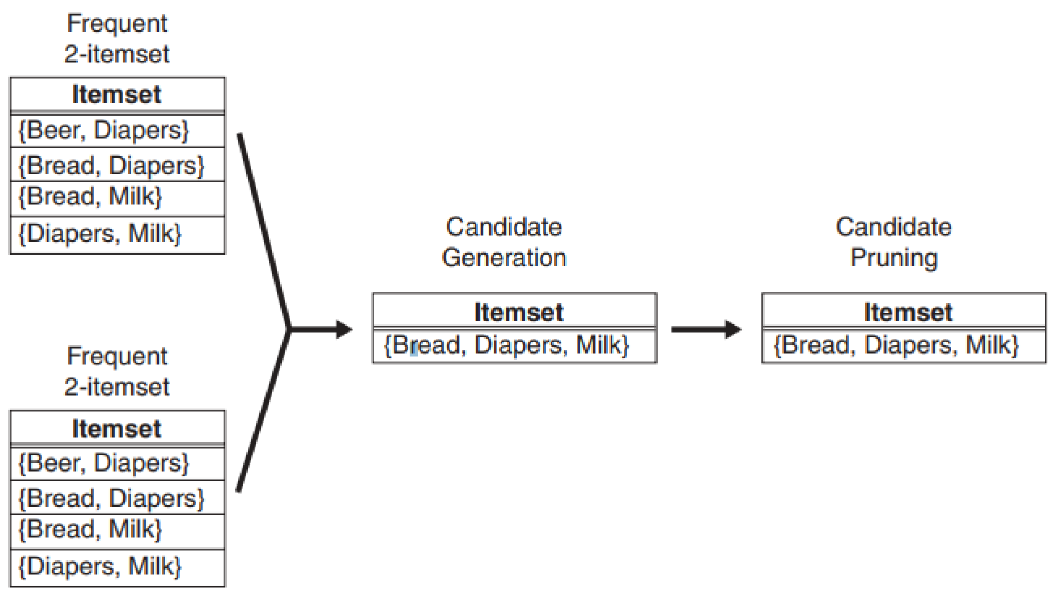


Figure 18. Candidate generation and pruning. (Tan et al. 2006)

There is no need to merge {Beer,Diapers} with {Diapers,Milk} as the same candidate would be generated by merging {Beer, Diapers} with {Beer, Milk} instead in case it would be a viable candidate (Tan et al. 2006).

4.3.1.2 Support Counting

The frequency of every candidate itemset that has survived the pruning step of the apriori-gen is counted in the support counting step. One way of counting the frequency is to compare each transaction against every candidate itemset and to update the support counts of the candidates contained in the transaction, but this approach is computationally expensive (Tan et al. 2006).

The apriori algorithm uses a more efficient way for support counting with the help of a hash tree structure. All candidate itemsets are stored in different buckets in a hash tree. Using this structure, each itemset in a transaction is hashed to their appropriate buckets and are only compared to the candidate itemsets in the same bucket. This way, there is no need to compare each itemset in the transaction to every candidate itemset. Figure 19 shows an example of this structure with candidate 3-itemsets. Each node in the structure uses the hash function $h(p) = p \bmod 3$ to determine the node to be followed. All the candidate itemsets are stored in the leaf nodes of the hash tree. To update the support counts of the candidate itemsets, the tree has to be traversed in a way that all the leaves containing candidate itemsets belonging to a transaction are visited at least once (Tan et al. 2006).

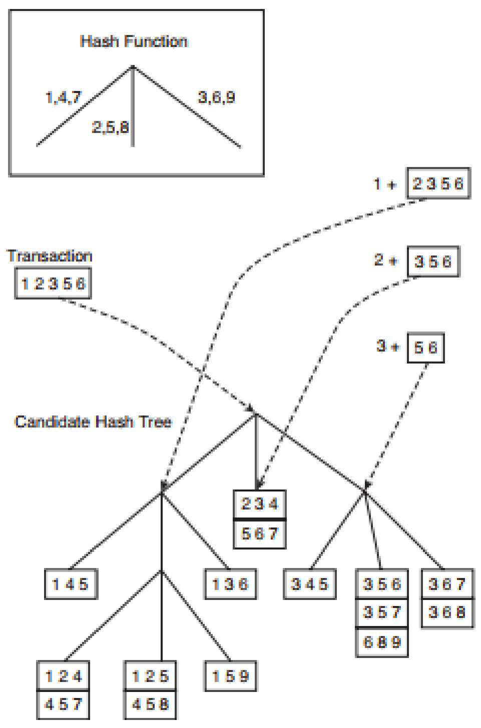


Figure 19. Hashing a transaction. (Tan et al. 2006)

Consider the transaction with items {1,2,3,5,6} in Figure 19. Items 1,2 and 3 will be hashed differently at the root node so that item one is hashed to the left child, 2 to the middle and 3 to the right. On the next level, the transaction is hashed on the next item. This process is continued until the leaf nodes are reached. The candidate itemsets found at the leaf nodes are compared against the transaction and if a candidate itemset is a subset of the transaction, the support counts are updated accordingly.

4.3.1.3 Complexity

There are multiple factors that have an impact on the computational complexity of the apriori-algorithm. As frequent itemsets are determined by the support threshold, lowering the threshold will usually result in more frequent itemsets. This will in return have a negative impact on the performance of the algorithm, as more itemsets has to be generated and counted. When the maximum size of itemsets is increased, the algorithm has to pass over the dataset more times (Tan et al. 2006).

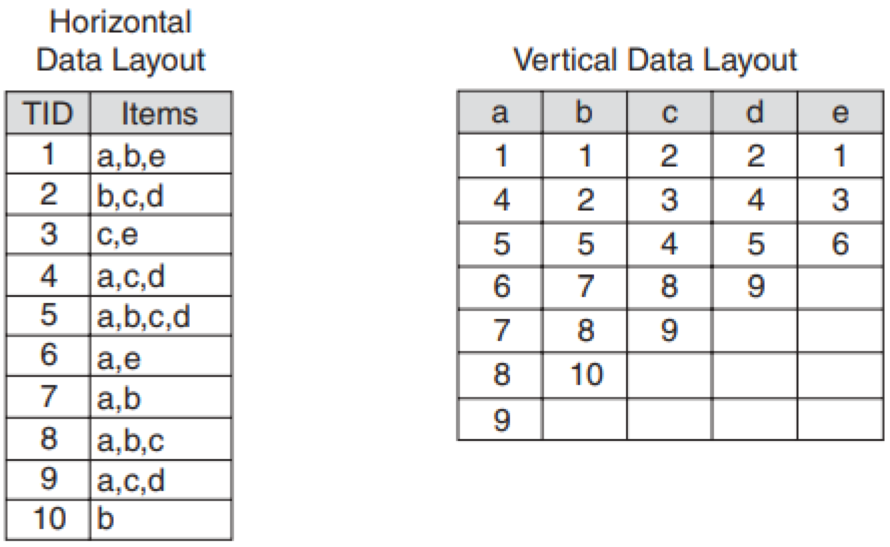
With increasing number of items, more space will be required to store the support counts of the items and if the frequent items increase with more items, the complexity will increase as more candidate itemsets are generated. Another factor that affects the performance is the amount of transactions, as the algorithm has to do more passes over the dataset (Tan et al. 2006).

The performance of the Apriori algorithm is also affected in two ways by the transaction width. First, with an increasing maximum size of itemsets more candidate itemsets must be evaluated. Second, as the transactions get larger the transactions will contain more itemsets (Tan et al. 2006).

4.3.2 ECLAT-ALGORITHM

The ECLAT-algorithm (Equivalence Class Transformation) was first introduced in 1997 by Zaki, Parthasarathy, Li and Ogihara. The algorithm takes a different approach to finding association rules compared to the Apriori algorithm. The Apriori algorithm takes a breadth-first approach to mining itemsets, whereas the ECLAT-algorithm is a depth-first search algorithm.

The transaction data set is structured differently than in the Apriori algorithm. Instead of storing the data in a horizontal manner, the data is represented in a vertical layout. This is seen as a TID-list, as it is a list of transaction identifiers. Each item is represented by a list of all the transactions it belongs to, as illustrated in Figure 20.



Horizontal Data Layout

TID	Items
1	a,b,e
2	b,c,d
3	c,e
4	a,c,d
5	a,b,c,d
6	a,e
7	a,b
8	a,b,c
9	a,c,d
10	b

Vertical Data Layout

a	b	c	d	e
1	1	2	2	1
4	2	3	4	3
5	5	4	5	6
6	7	8	9	
7	8	9		
8	10			
9				

Figure 20. Vertical and horizontal layout. (Tan et al. 2006)

The idea behind the Eclat-algorithm is that the support can be counted by intersecting the candidate itemsets. Given transactions t(X) and t(Y) for two frequent itemsets X and Y, it the intersection can be written as follows:

$t(X,Y) = t(X) \cap t(Y)$.

As an example, for TID-sets A = 1345 and B = 2456, the support of AB can be calculated by intersecting the TID-sets, 1345 ∩ 2456 = 45. In this case, the support for the example is 2. ECLAT intersects TID-sets only if the frequent itemsets share a common prefix. It traverses the tree in a depth-first manner by processing a group of itemsets that have the same prefix, also called a prefix equivalence class (Zaki & Meira, 2013).

Figure 21 illustrates an example of the ECLAT-algorithm with a minimum support of 3. The initial prefix equivalence class is {(A,1345), (B,123456), (C,2456), (D,1356), (E, 12345)}. ECLAT intersects A with the other TID-sets to obtain the TID-sets AB, AC, AD and AE. AC is infrequent and pruned. The frequent itemsets form a new prefix equivalence class, {(AB,1345), (AD,135), (AE,1345)} which is then recursively processed. All branches are processed in a similar fashion, and the entire tree is illustrated in Figure 21, with infrequent TID-sets marked gray. The Eclat-algorithm itself is described in Figure 22.

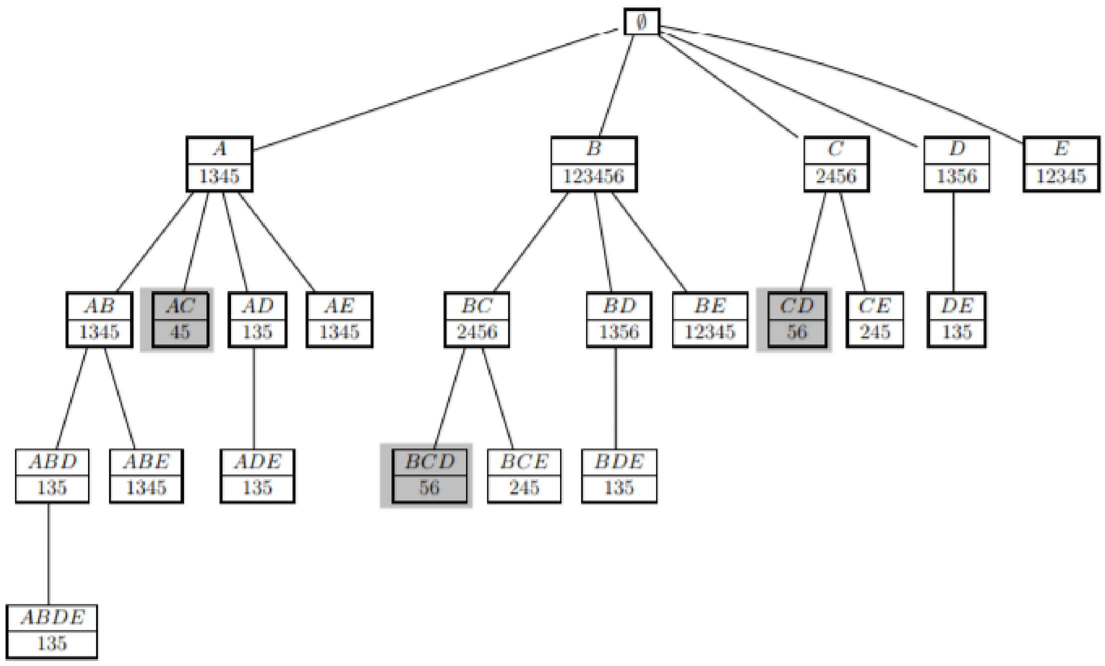


Figure 21. Eclat-algorithm, example (Zaki & Meira, 2013).

```
// Initial Call: $\mathcal{F} \leftarrow \emptyset, P \leftarrow \{\langle i, \mathbf{t}(i)\rangle \mid i \in \mathcal{I}, |\mathbf{t}(i)| \geq minsup\}$
ECLAT ($P$, $minsup$, $\mathcal{F}$):
1 foreach $\langle X_a, \mathbf{t}(X_a)\rangle \in P$ do
2     $\mathcal{F} \leftarrow \mathcal{F} \cup \{(X_a, sup(X_a))\}$
3     $P_a \leftarrow \emptyset$
4     foreach $\langle X_b, \mathbf{t}(X_b)\rangle \in P$, with $X_b > X_a$ do
5         $X_{ab} = X_a \cup X_b$
6         $\mathbf{t}(X_{ab}) = \mathbf{t}(X_a) \cap \mathbf{t}(X_b)$
7         if $sup(X_{ab}) \geq minsup$ then
8             $P_a \leftarrow P_a \cup \{\langle X_{ab}, \mathbf{t}(X_{ab})\rangle\}$
9     if $P_a \neq \emptyset$ then ECLAT ($P_a$, $minsup$, $\mathcal{F}$)
```

Figure 22. The Eclat-algorithm. (Zaki & Meira, 2013)

The complexity of the Eclat-algorithm is hard to characterize, as it is largely dependent on the size of the intermediate TID-sets. It is thereby possible to improve the algorithm by shrinking the size of the intermediate TID-sets. The dEclat-algorithm keeps track of the difference in each TID-set, called diffset, instead of keeping track of the full TID-set.

This is viable as it is possible to obtain a diffset from the diffsets of the subsets. The support of a candidate itemset can be calculated by subtracting the diffset size from the support of the prefix itemset. The dEclat-algorithm is illustrated in Figure 23, with infrequent itemsets marked as gray. To process candidates with A as a prefix, the dEclat-algorithm calculates the diffsets for AB, AC, AD and AE. In this example the diffset of AB = ∅, and AC = 1,3. The corresponding support values are sup(AB) = 4-0 = 4 and sup(AC) = 4-2 = 2, meaning that AC is pruned. The rest of the branches are processed in a similar way.

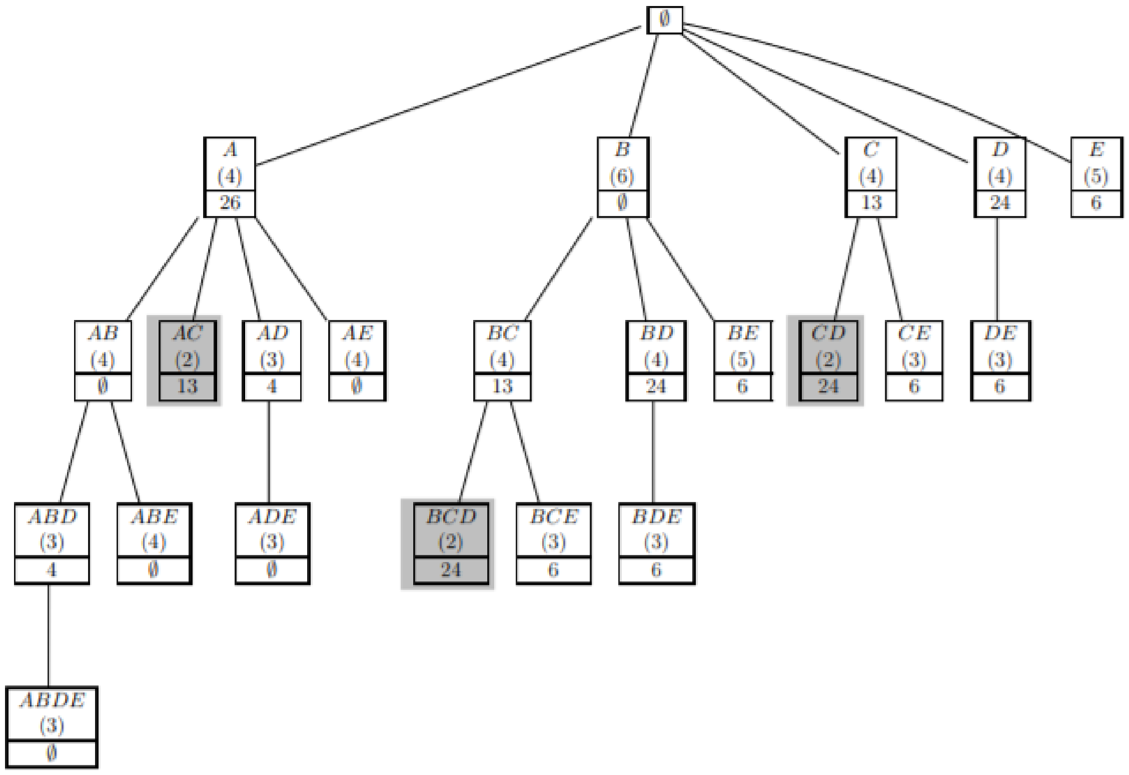


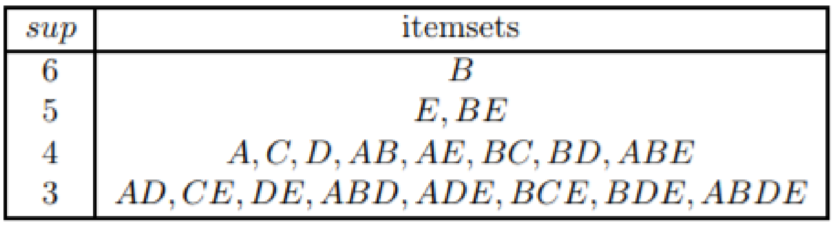
Figure 23. The dEclat-algorithm (Zaki & Meira, 2013).

4.3.3 FP-GROWTH

The FP-Growth algorithm finds frequent itemsets by first generating a special data structure called frequent pattern tree (FP-tree). Each node in the tree is labeled as a single item in the transaction dataset. Each child node will represent a different item. Each node in the tree will store the support count among the path from the root for the itemset. First, all items will be ordered in descending order based on their support counts. Each transaction will also be ordered based on the initial support-order and items that are under the support threshold will be eliminated (Zaki & Meira, 2013).

The FP-tree is first initialized with a null node. Then, each itemset is inserted into the tree and support counts for each node in the itemset is increased along the correct path in the FP-tree. If the next itemset shares items with the existing nodes in the tree, it will follow the path as long as the items remain the same. For the remaining items in the new itemset, new nodes will be added to the tree with initial support counts of 1. When all transactions have been added to the FP-tree, the FP-tree is considered complete. As the support counts are calculated for all single items, and the inserted itemsets are ordered based on this order, the tree will be as compact as possible (Zaki & Meira, 2013).

Zaki & Meira (2013) have illustrated the algorithm with an example. Consider the itemsets in Table 4. The sorted item order for the dataset is B(6), E(4), A(4), C(4), D(4). Each itemset is ordered according to this order and inserted to the tree. Figure 24 illustrates this process one transaction at a time.



sup	itemsets
6	B
5	E, BE
4	$A, C, D, AB, AE, BC, BD, ABE$
3	$AD, CE, DE, ABD, ADE, BCE, BDE, ABDE$

Table 4. Frequent itemsets with min support = 3. (Zaki & Meira, 2013)

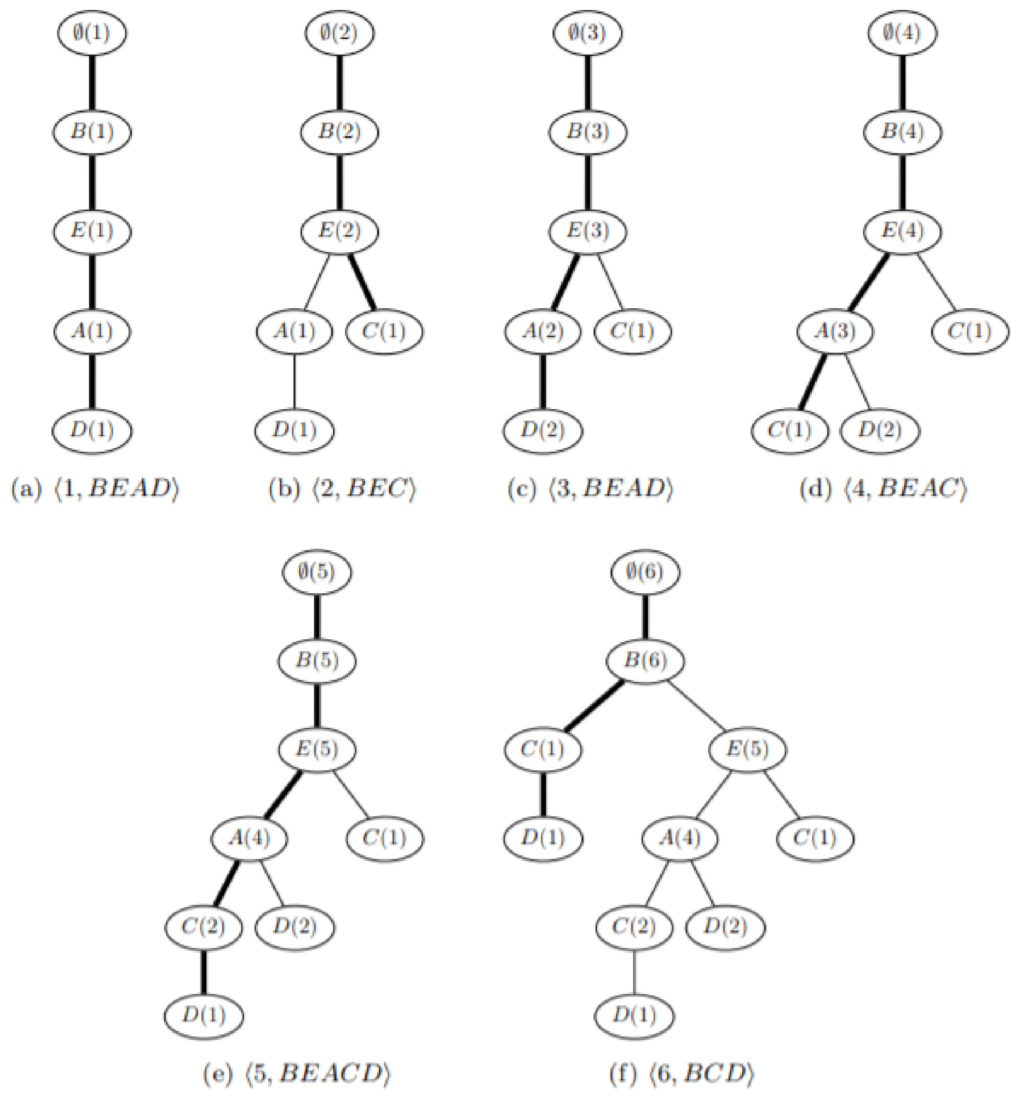


Figure 24. Inserting transactions to the FP-tree. (Zaki & Meira, 2013)

Once the FP-tree has been generated, it represents the original dataset with support counts in a tree data structure. The FP-tree can then be mined for frequent itemsets with the FPGrowth-method, illustrated in Figure 25. FPGrowth creates a new projected FP-tree from each item, but in increasing support count. Given the initial FP-tree in Figure 24, there are 3 different paths for item D. These paths, excluding the last item D, are inserted to the tree. The count of the occurrence of item D on the path in question is also calculated. The projected FP-tree for item D is illustrated in Figure 26.

```
// Initial Call: $R \leftarrow$ FP-tree($\mathbf{D}$), $P \leftarrow \emptyset$, $\mathcal{F} \leftarrow \emptyset$
FPGrowth ($R$, $P$, $\mathcal{F}$, $minsup$):
Remove infrequent items from $R$
if IsPath($R$) then // insert subsets of $R$ into $\mathcal{F}$
    foreach $Y \subseteq R$ do
        $X \leftarrow P \cup Y$
        $sup(X) \leftarrow \min_{x \in Y}\{cnt(x)\}$
        $\mathcal{F} \leftarrow \mathcal{F} \cup \{(X, sup(X))\}$
else // process projected FP-trees for each frequent item $i$
    foreach $i \in R$ in increasing order of $sup(i)$ do
        $X \leftarrow P \cup \{i\}$
        $sup(X) \leftarrow sup(i)$ // sum of $cnt(i)$ for all nodes labeled $i$
        $\mathcal{F} \leftarrow \mathcal{F} \cup \{(X, sup(X))\}$
        $R_X \leftarrow \emptyset$ // projected FP-tree for $X$
        foreach $path \in$ PathFromRoot($i$) do
            $cnt(i) \leftarrow$ count of $i$ in $path$
            Insert $path$, excluding $i$, into FP-tree $R_X$ with count $cnt(i)$
        if $R_X \neq \emptyset$ then FPGrowth ($R_X$, $X$, $\mathcal{F}$, $minsup$)
```

Figure 25. FPGrowth-method. (Zaki & Meira, 2013)

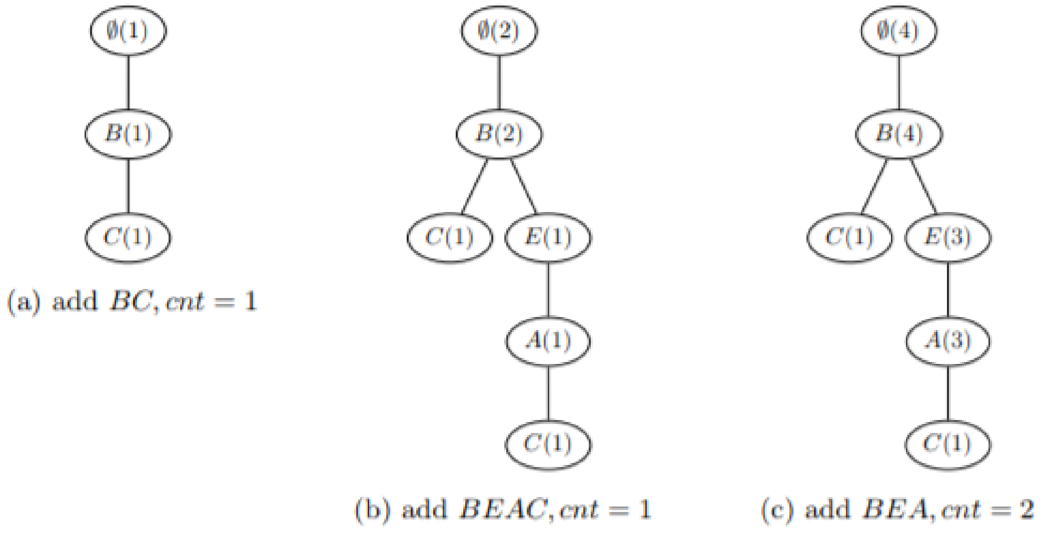


Figure 26. Projected FP-tree. (Zaki & Meira, 2013)

After processing the FP-tree, and after removing the items below minimum support count (item C), we have the itemsets for prefix D. This results in a single path B(4)–E(3)–A(3). By splitting this to all subsets and prefixing them with D, the frequent itemsets DB(4), DE(3), DA(3), DBE(3), DBA(3), DEA(3) and DBEA(3) are obtained. The frequent itemsets for the other prefixes can be generated in the same way (Zaki & Meira, 2013). The tree projection for the other items is illustrated in Figure 27.

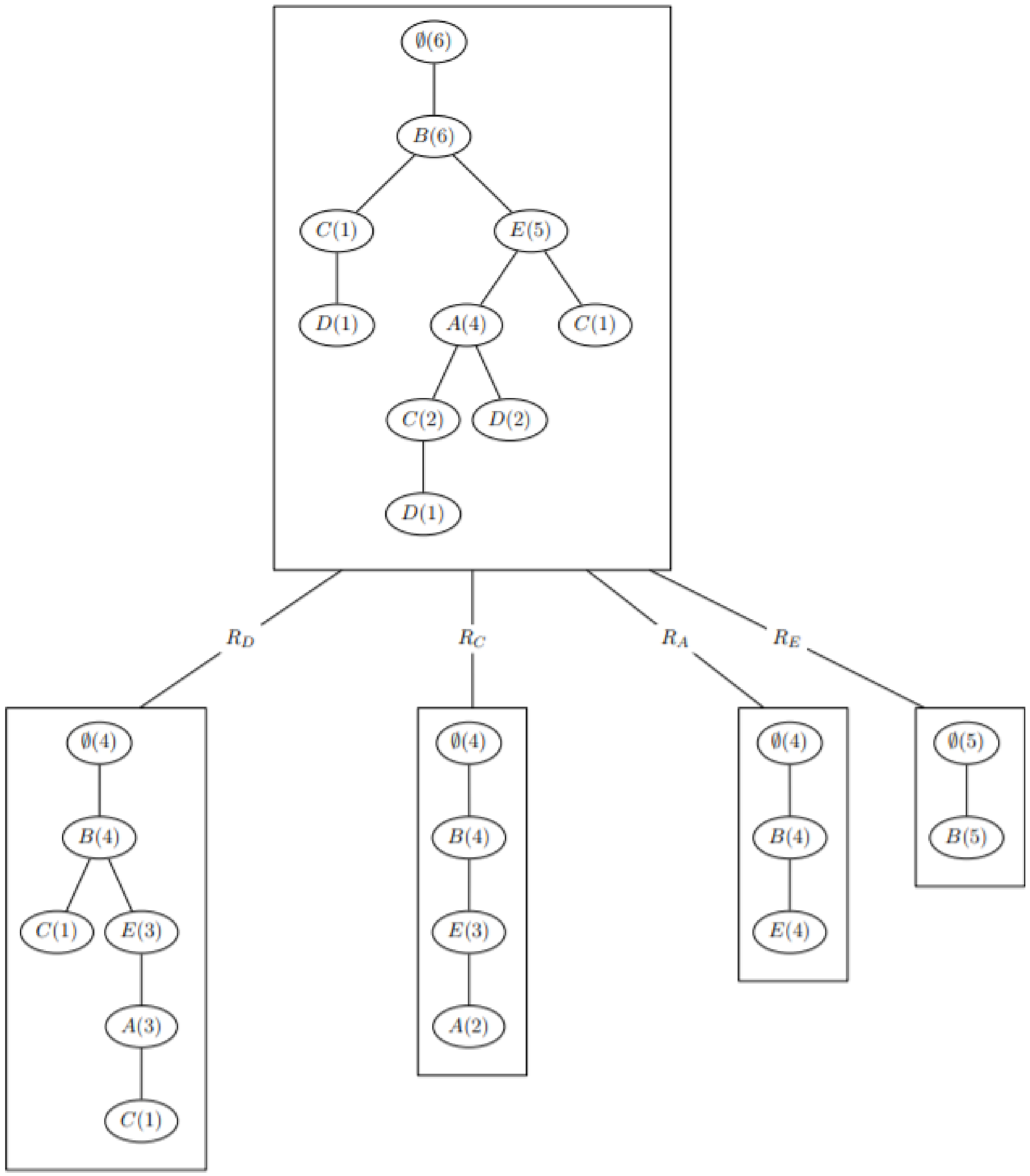


Figure 27. FP-tree projection. (Zaki & Meira, 2013)

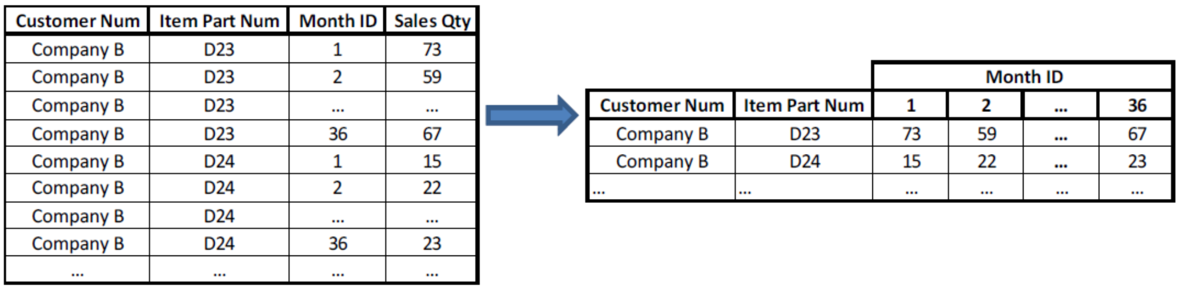
4.4 ASSOCIATION RULE GENERATION

When the frequent itemsets have been calculated, it is possible to obtain association rules from the collection of the frequent itemsets. All itemsets are iterated to calculate the confidence of various rules that can be derived from the itemset. Given a frequent itemset Z, the association rules are in the form X → Y, where X and Y are subsets of Z. To calculate the confidence of a rule, it can be done with the formula *sup*(Z) / *sup*(Y) (Zaki & Meira, 2013).

Taking the frequent itemset ABDE(3) from Table 4 as an example, and using 0.9 as a minimum confidence threshold, the set of antecedents are ABD(3), ABE(4), ADE(3), BDE(3), AB(3), AD(4), AE(4), BD(4), BE(5), DE(3), A(4), B(6), D(4), E(5). The first subset is ABD, and the confidence for the rule ABD → E is 3/3 = 1.0, so the rule is strong. The next subset is ABE and the confidence for the rule ABE → D is 3/4 = 0,75, so the rule is discarded. We can therefore remove all subsets of ABE from the antecedents, so the updated antecedents are ADE(3), BDE(3), AD(4), BD(4), DE(3) and D(4). Following this logic we can extract all strong rules from the list of frequent itemsets (Zaki & Meira, 2013).

4.5 OTHER APPROACHES

Tan & Lau (2013) have done a study on time-series clustering as an alternative to regular market basket analysis. They argue that market basket analysis is not always the best alternative, especially when the dataset is large with many products. This is because of long processing times and many rules without much insight. Instead, clustering on transactional data formatted as time-series data can provide better insight. The dataset is transformed as follows: Each month gets an identifier and then represented as a column in the data matrix. This is best illustrated in Figure 28.



Customer Num	Item Part Num	Month ID	Sales Qty
Company B	D23	1	73
Company B	D23	2	59
Company B	D23	...	...
Company B	D23	36	67
Company B	D24	1	15
Company B	D24	2	22
Company B	D24	...	...
Company B	D24	36	23
...	...	...	...

		Month ID			
Customer Num	Item Part Num	1	2	...	36
Company B	D23	73	59	...	67
Company B	D24	15	22	...	23
...	...	...	...	...	...

Figure 28. Transactional data formatted as time-series data. (Tan & Lau, 2013)

Using the k-means algorithm, they managed to get useful patterns of products bought together. This can be used for cross-selling and better inventory control. One of the benefits of this approach is the drastically reduced size of the data. Since market basket analysis is widely used, many problems can be tackled with this approach instead. Further suggested research includes applying other (more advanced) clustering algorithms with this approach.

Videla-Cavieres & Ríos (2014) have presented an approach for market basket analysis based on graph mining techniques. Their method is useful when the data is huge and scattered and common techniques fail. The graph is generated by connecting each transaction with corresponding products. The graph is then transformed to a product-to-product weighted network. With the help of community detection algorithms frequent itemsets can be discovered.

According to Baer & Chakraborty (2013) there are three approaches for product affinity segmentation. Customers can be segmented by demographic or other transaction variables which can then be combined with product-level purchase data. Market Basket Analysis using POS data can be used to understand what products are bought together, and finally product type data can be directly used in clustering algorithms to find affinity segments. They have applied the “doughnut” clustering method to generate customer clusters based on purchase patterns. This method is useful when the goal is to find out which customers have similar purchase patterns and can be used for more effective marketing.

5 ANALYSIS

This chapter will give an overview of some of the different tools and technologies that are relevant to data warehousing and market basket analysis. The chapter will however only give a brief introduction to some of the options that are available, as the focus of the thesis is on the process and models instead. Followed by an introduction to the tools, the general architecture of the DW/MBA system will be explained.

5.1 CLOUD TECHNOLOGIES

In recent years, cloud computing has emerged as an attractive alternative to traditional on-premise solutions. Common cloud services include Amazon AWS and Microsoft Azure. According to Carroll et al. (2011) and Rittinghouse & Ransome (2009), there are some obvious benefits in cloud computing, some of which are:

- Scalability
- Flexibility
- Reduced implementation and maintenance costs, cost efficiency
- Availability of high-performance applications
- Fast start up

The usage of cloud computing is however dependent on the business at hand, and every project should be evaluated independently. As an example, Aljabre (2012) has pointed out that cloud services may still lack some features. Cloud services might also raise problems with the security of confidential data.

5.2 ETL-TOOLS

There are various ways of handling the ETL-process for the data warehouse solution. First, there are many different ETL-tools available on the market, both open-source and commercial. Second, it is completely viable to handle the ETL-process by writing own code in the form of stored procedures in the database or by writing scripts for example in Python or PowerShell.

If using SQL Server as the database, SSIS is a good option for an ETL-tool. It is offered by Microsoft and included in the SQL Server licence. SSIS offers great integration with SQL Server, making it a good choice. Gartner (2018) has also listed Informatica PowerCenter and the open source-tool Talend as other good options rated by customers. Another common open-source tool is Kettle by Pentaho.

As mentioned earlier, it is also possible to opt for a solution without an ETL-tool. One option is to use plain SQL to write all transforms and loads. The benefit of this approach is a simpler architecture as no other tools are needed, and as SQL is universal and well known it is easier for new developers to work on the data warehouse as there is no need for knowledge of another tool. Quite often, it is also easier to write the transforms and loads in plain SQL compared to using an ETL-tool. The drawback of this solution is the lack of visual aid, and complex ETL-processes can become very hard to maintain and understand. Writing the ETL in a scripting language can give greater control and flexibility compared to plain SQL, but requires more expertise as it requires knowledge in another language in addition to SQL.

The choice of an ETL-tool is largely dependent on company preferences regarding technology, budget and architectural preferences. If the data warehouse is small and the ETL is fairly simple, a SQL-based ETL-pipeline will be perfectly fine. Larger and more complex solutions on the other hand may gain significant advantage from the visual aid provided by ETL-tools.

5.3 DATABASE ENGINES

Today, there are multiple database engines to choose from. First of all, cloud solutions will have different options compared to a traditional data-center or on-premise approach for the database. In the traditional approach, there are open-source, free and paid services. From the open-source possibilities, the most common database engines are MySQL and PostgreSQL. From the commercial options Microsoft SQL Server and Oracle Database are commonly used. Microsoft also offers a free version of SQL Server, called the Express version. The SQL Server Express Edition is completely suitable for small enterprise usage, but it has some restrictions, as an example the maximum database size limit is 10GB (Microsoft, 2019). SQL Server uses a specific language that is based on SQL, called T-SQL.

As I am used to work with SQL Server, the architecture will be planned with SQL Server in mind and code will be written in T-SQL. This can however be changed based on the organization in question depending on preferences and existing architecture.

5.4 ANALYTICS OPTIONS

There are multiple tools and technologies that can be used for performing market basket analysis. Examples of the tools that can generate association rules are WEKA and Rapidminer, but it is also common to use programming/scripting languages for the task. For example, R and Python have ready libraries for this task. A common implementation is to use R, and there are multiple tutorials available on the internet. As a bonus, Microsoft has added support for R in SQL Server from version 2016 onward, which means that the scripts can be embedded in stored procedures inside the data warehouse. This will make the entire solution even cleaner and help in maintenance as all code can be found and ran in one place. If using another database, the R script can be triggered in other ways.

To perform market basket analysis in R, the easiest way is to import the “arules”-package. The “arules”-package contains implementations for both eclat and apriori and can be performed by calling the eclat()- and apriori()-functions (Hahsler et al. 2019)

5.5 RETAIL DATA PIPELINE

This chapter introduces the artifact that was created to answer the research questions. In this case, the artifact is the entire architecture and proposed system. It is designed with a data warehouse that works as source for the MBA-integration. The architecture is also designed with automation in mind. Another option would naturally be to obtain data from source systems and perform manual MBA whenever needed. This option would be perfectly fine if the organization is only looking to do occasional ad-hoc analyses, and does not need automated analytics. The proposed architecture will however also work as a stable foundation for future analytics needs, and is deployable on both cloud and on-premise platforms.

5.5.1 ARCHITECTURE

Based on the literature review and the scope of this thesis, it seems that the best approach would be to build a two-layered data warehouse architecture as a base if the company doesn't already have a functional data warehouse solution. By using a data warehouse as a basis for the architecture, it allows for much better scalability and easy development of other analytical solutions. An option would be to build an automated market basket analysis solution on the raw source data, but this solution will not support other analytics and BI needs in the future. Building the solution on raw source data runs a risk of becoming a nightmare to maintain if all subsequent analytical solutions are developed in a similar way.

In the proposed solution, it is also assumed that the company does not have an important need for data auditing and traceability, but rather wants to focus on analysis. A two-layered architecture will deliver results much quicker and therefore also hold a much smaller budget compared to a three-layered architecture. This architecture means that the data warehouse layer itself should be based on dimensional modeling. The complete architecture is illustrated in Figure 29 with the following layers:

1. Source systems: The source data may be available e.g. in OLTP-databases or provided as flat files.
2. Staging area: The staging area consists of database tables that mimic the structure of the source tables and/or source flat files.
3. Data Warehouse layer: This is where the actual data is stored. The structure is

based on the Kimball approach, meaning that the layer is based on dimensional modeling.

4. Analytics layer: In this layer MBA is performed. Any future BI/Analytics will be built in this layer.
5. Presentation/End users: The data is delivered in a usable and easily understandable format for end users in a way that can help in decision making.

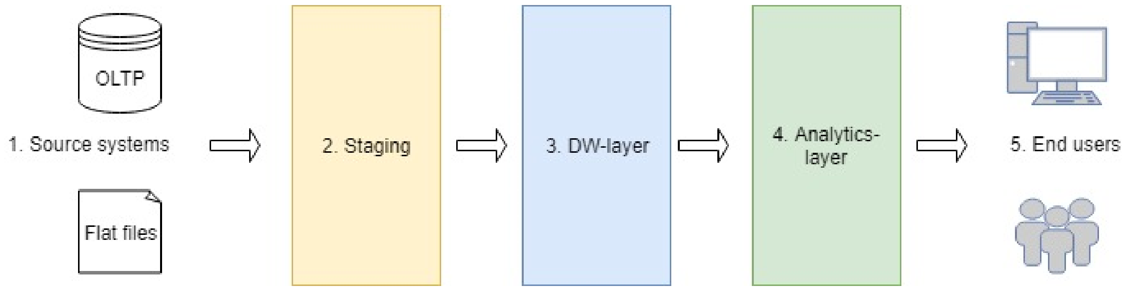


Figure 29. The proposed architecture.

An example of a retail POS-system is illustrated in Figure 30. It is fairly simple to extract the needed data from the source tables into the staging area and load the data into a star schema. In another situation, the data might have to be delivered as flat files, and in this case the staging tables are based on the columns in the flat file(s).

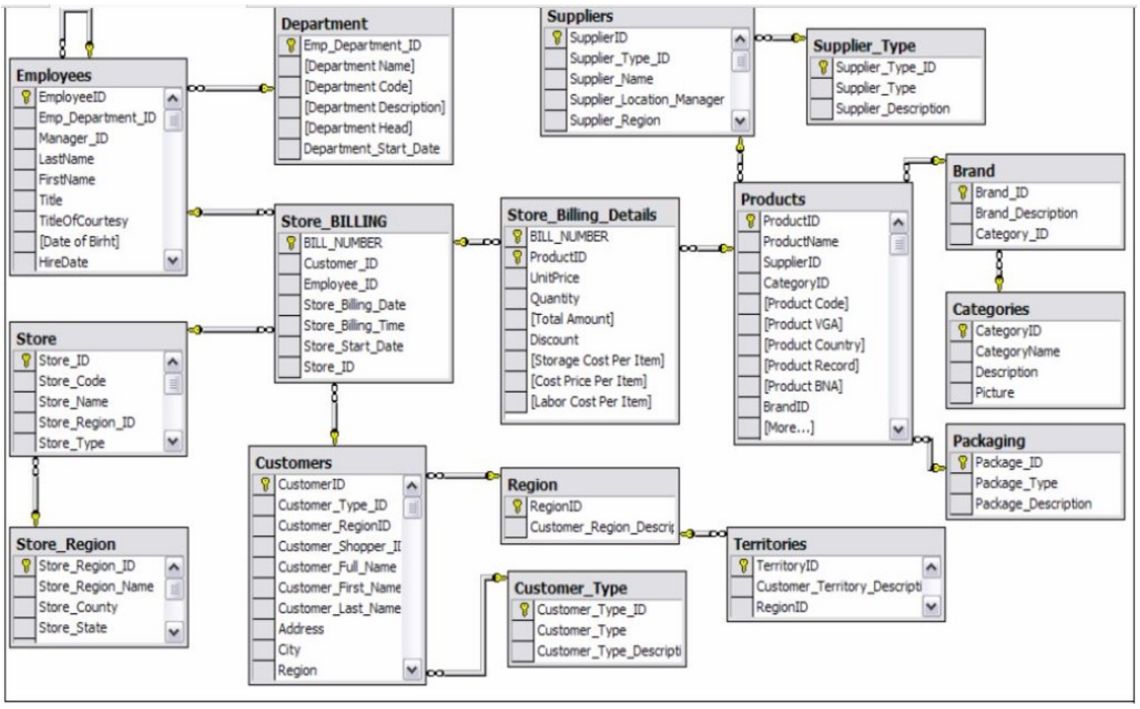


Figure 30. Example ER-diagram of a retail sales business.

Following agile and scrum philosophy, the development should start with the minimum viable product (MVP). The MVP is an actual product with least effort that can be offered to customers to be further observed (Agile Alliance, 2019). In this case, the star schema is the first thing that should be developed, and it is fairly quick to do. Taking the business goals into account, the dimensional model should include at least the fact table, the product dimension, the customer dimension and the date dimension. Figure 31 illustrates the star schema for this purpose.

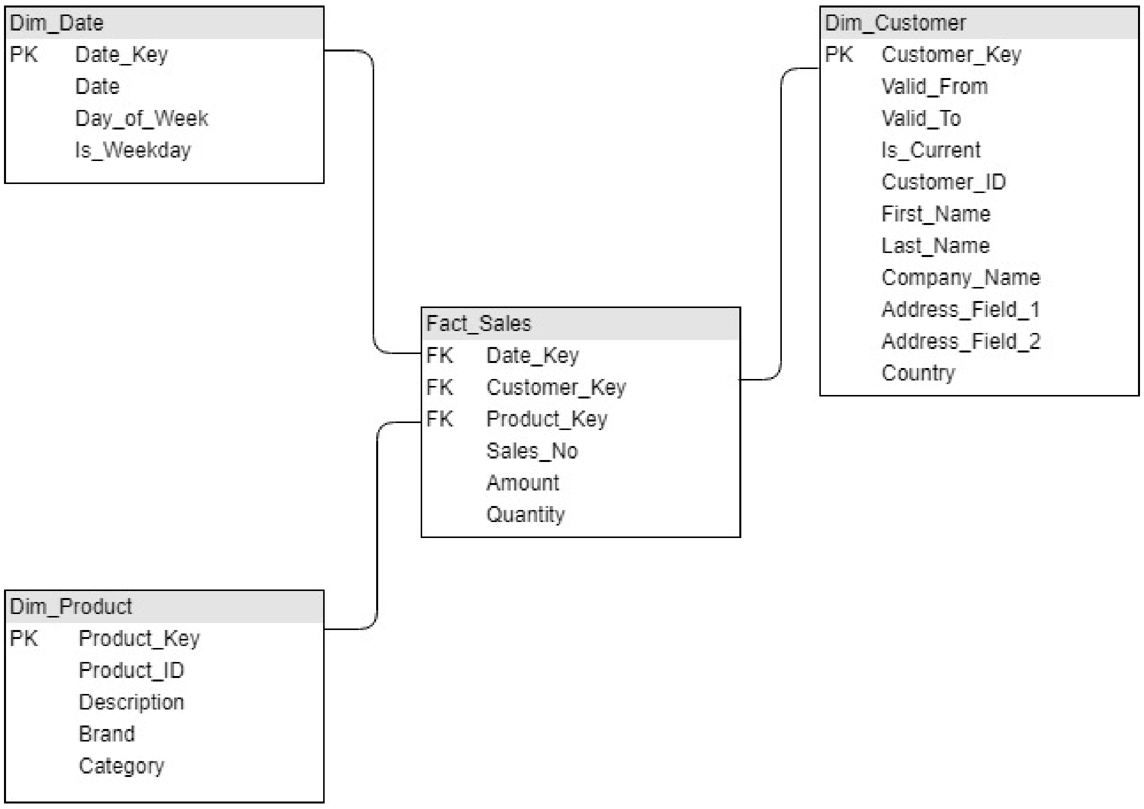


Figure 31. Proposed Star Schema.

Dim_Customer: Customer dimension. As customer data will change over time and there might be need to track changes at some point, SCD Type 2 would be a great option for this dimension as it allows track of the history while still staying simple. Valid_From is a date field that marks when the row is effective, Valid_To marks the expiration date, and Is_Current indicates the current row.

Dim_Date: Date dimension. The Date dimension can be modeled as a Type 0 dimension, as it will most likely not change. It includes the date in a date data type which can be used for date functions and more fields can easily be added as needed.

Dim_Product: Product dimension. The product dimension can be modeled as a Type 1 dimension, unless the organization has specific needs for a more advanced type. This means that if the product description changes, the old information will be destroyed.

Fact_Sales: Fact table for sales. The Fact table stores the product sold and is obviously at the transaction grain. The Fact table will also include fields for quantity and total amount so that the Fact-table can be aggregated logically.

The creation of the star schema is quite straightforward. The entire schema can be generated with “CREATE TABLE”-statements. The data types and lengths can be changed according to specific business needs. The staging layer also has to be created, and it should follow the table definitions in the source tables as discussed earlier in the thesis. A quick way is to use a visual tool, like SSMS for SQL Server, to generate table scripts of the needed tables.

5.5.2 ETL-PROCESS

The entire ETL process consists of the following steps in their specific order:

1. Extract and load raw source data into staging tables
2. Transform and load dimension tables
3. Transform and load the fact table
4. Run analytics

Depending on the source database engine, there might be different needs to import the data into the SQL Server staging tables. One option is to create a linked server, and another is to use SSIS for the import task. If the linked server route is an option, it allows for the ETL process to be built entirely using stored procedures which creates a simple and easy solution. The first step in the ETL process is thereby to load the wanted data into the staging tables. For all tables except for the fact-source, it is best to load all rows into the staging tables. For the fact-source, a good option is to filter the extraction based on the time frame. If the ETL-process is to be ran once a day, then only sales for the day in question should be loaded into the staging table. Depending on the source, it might however only be feasible to load all data and do the actual filtering when loading the data into the data warehouse itself.

The next step is to load the star schema. Due to the structure of a star schema, mainly because of the foreign key constraints in the fact table, the dimensions have to be loaded before the fact table. The Date dimension should be loaded only once manually, and it can be accomplished by writing T-SQL scripts using any of many guides available on the web. Another option is to download e.g. a csv-file that contains the needed data. This approach can be good especially if banking holidays are of importance. The rest of the dimension tables can be loaded with stored procedures.

When loading the data into the data warehouse layer, transformations are almost always necessary in some form. Data in source systems and files are usually of varying data types, and this should be taken into account in the ETL-process. This means that the data and data types should be transformed when loaded from the staging tables into the data warehouse. The data types should be conformed in the data warehouse to keep consistency in the data and the model. Take dates as an example, they can be stored as integers, dates, datetimes and strings. Strings can be further stored in varying formats, e.g. 'DD/MM/YYYY' or 'YYYY--MM-DD' which means that the string needs to be parsed and transformed to the correct data type. Another example is of currency amounts, as they can be of different precision and scale e.g. 122,24600 and 122,25.

Another issue arises when different source systems store data of the same things. The data warehouse should not contain duplicate information, it should instead store "a single truth" to which all business areas and processes reference. When conforming the data, focus should be put on field names and naming conventions, so that they are easy to understand and same names actually mean the same thing.

The product dimension should be loaded as follows: First, the needed data of the products should be gathered from the relevant staging tables and joined together. Next, the needed data cleansing and transforming should be performed so that the data is in good shape. Next, the transformed data should be left joined to the existing rows in the product dimension on the product id attribute to check whether a specific product already exists. If the product exists, the attributes should be updated accordingly as the dimension is a type 1 slowly changing dimension. If the product is new, the row is inserted into the dimension. An easy option for surrogate key generation is to set a new IDENTITY-constraint on the table column Product_Key. This way every new row will get a new integer value.

As the customer dimension is a type 2 slowly changing dimension, it will require some additional logic for loading. The rows that do not exist in the source data should be terminated, meaning that Is_Current should be set to 0 and Valid_To should be set to the current date. Next new rows that do not exist in the dimension should be added with their correct attributes. For records that have changed, the existing rows need to be terminated, and new rows will be added.

When the dimensions have been loaded, it is time to load the fact table. First, the needed facts should be gathered from the correct staging table. In the retail case, if the grain is not on product sold, it should be changed. The fact data should then be joined to the corresponding staging tables to get the customer id and product id. This dataset should then be joined to the dimensions loaded in the previous step to get the Customer_Key and Product_Key attributes. The enriched dataset should then be loaded into the fact table. The date key can be loaded as the transaction date converted to integer data type.

At this point, the star schema has been loaded and the data is ready for use. It is easy to build different types of validations to run after the ETL-process to make sure that the data is loaded correctly. Some of the easy options include calculating row counts and comparisons to previous load dates.

To actually develop an effective data warehouse solution, the ETL has to be automated. Running the ETL-process manually is cumbersome and ineffective, as the goal is that the ETL-process will succeed most times. This means that starting of the ETL-process should be scheduled by using a scheduler of some sort. It is of course also possible to trigger the ETL-process as a result of a specific event, e.g. when source files have arrived. Common options for the scheduling of the job is to use SQL Server agent jobs, Windows Task scheduler or a scheduler provided by the cloud service if opting for a cloud solution. The ETL-process should then run loading processes in the correct order, and only starting a task if its prerequisite has completed. If not using an ETL-tool, one option is to use a “master”-procedure that runs the subtasks in the correct order.

Depending on the needs of the data warehouse, the ETL scheduling should be set to run on an interval, e.g. daily or monthly. The interval is naturally dependent on the analytical needs of the business, as if the business only needs to run analytics and reports on a monthly basis, running the ETL-process monthly will suffice. In other cases the business will want the newest data available every day, and sometimes there might be a need to load new data into the data warehouse many times a day or even do real-time updating. If querying OLTP-systems, it makes sense to schedule the ETL-process for times when the transactional load is lower, e.g. during the night.

A good ETL-process also consists of monitoring. SQL Server Agent allows for decent monitoring capabilities. Whenever a job completes or fails, it can be configured to send an email. Besides this, the job history is logged, so events can be tracked. SQL Server Agent also has integration with SSIS-packages which helps in monitoring. SQL Server also supports SSIS-reports, which are quite helpful for tracking statistics and errors in executed packages. Other tools, cloud services and RDBMS have similar capabilities, so the actual monitoring is heavily dependent on the technology at hand. It is obviously also possible to build own monitoring solutions that can be customized for the data warehouse.

5.5.3 INTEGRATION TO MARKET BASKET ANALYSIS

Based on the literature review, it seems most logical to start by using the Apriori algorithm, as it is the most common and easy to implement in R. When the general architecture is built, it is fairly easy to run different analyses and algorithms on the data. Using the dimensional model as a source for analysis, it is easy to access the needed data to run the Apriori algorithm in R. The proposed architecture is based on SQL Server 2016 and onward, as the Machine Learning Services support R scripts inside stored procedures. The proposed architecture for the R integration is illustrated in Figure 32.

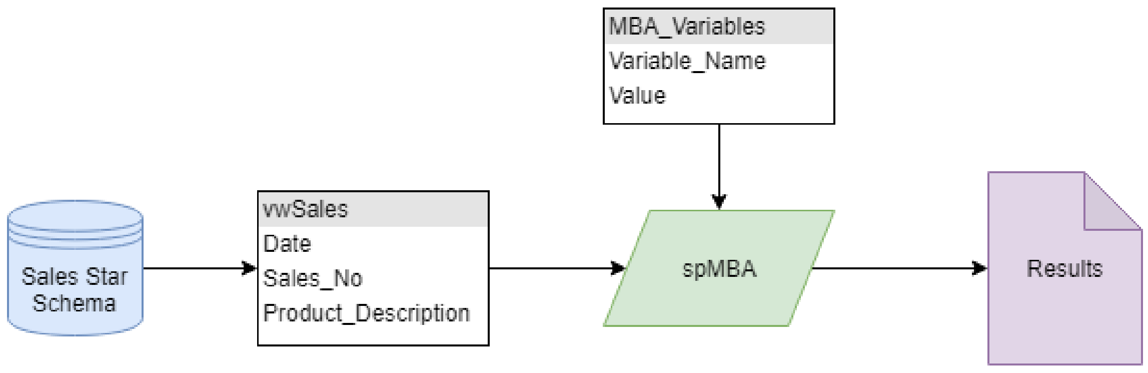


Figure 32. Proposed R integration.

All the data needed can be found from the sales star schema by joining the Fact_Sales to the Dim_Product dimension. To further control the dates in the analysis, Fact_Sales can be joined to the Dim_Date-table. Sometimes, there might also be a need to further restrict the data based on customer attributes, and this achieved by joining to the customer dimension. The integration can be simplified by using a view (vwSales in Figure 31) that joins the tables. The view joins the fact table with the dimension tables on their specific keys.

There are different ways how the data should be formatted for MBA in R. The common way is to format the data into the market basket format which is illustrated in Table 5 below. Another option is to format the data into a transaction object with R-functions. In this case the source data can be in basic format, meaning that one row has only one product.

Transaction_ID	Items
10001	A,C,D
10002	B,C,E
10004	A,B,C,E
10006	B,E

Table 5. Market basket format. (Modified from Gulalkari, 2016)

By querying the view with an optional WHERE-clause for date control, the data is ready for analysis. The WHERE-clause can naturally be used for any other type of filtering that is needed, e.g. removal of transactions that include specific items.

The stored procedure with the embedded R script (spMBA in Figure 32) will use the data from the view as input. If using another technology than Machine Learning Services for

SQL Server, the stored procedure is replaced with the R script itself and triggered in another way. The first step when setting up the automated market basket analysis is to configure the script. This will take a lot of trial and error as the support and confidence thresholds have to be determined. Additionally, it will require decisions on the data time interval on and how often the script should run, e.g. running the script monthly with the accumulated data from the past month. Next, depending on the performance of the Apriori-algorithm, it might be useful to test the analysis with another algorithm.

To make the integration more generic, some variables should be added to the architecture. By adding a table that holds variables (MBA_Variables in Figure 32), the script can be controlled. The stored procedure will be built with dynamic code, so that the script changes based on the variables. The benefit of this approach is that the script will require less modifications, as the behaviour of the script will change based on the variables in the control-table. As a starter, the following variables should be added:

- Minimum support
- Minimum confidence
- Data interval in days, meaning how old data should be used
- Algorithm to be used, in this case Apriori/Eclat.

Finally, the results should be delivered in a clean and readable way. One option is to format the results as HTML and send it as email. This way the results of the analysis will automatically appear e.g. monthly as an email and the business can do decisions based on this or do further analysis. Another option is to write the results back into the database, as this allows for historical storage and usage for multiple purposes. The results can also be saved in another format, e.g. .jpeg, and saved to disk. The results of MBA can be viewed in multiple ways, e.g. as a list of rules or as graphs. The presentation of data and results are however out of scope for this thesis.

5.5.4 EXPANDABILITY OF THE MODEL

The goal of the architecture is to also support expansion to other types of analytics. This means that new data from the sales process itself and from other processes should be easily integratable to the data warehouse so that further analysis can be performed. Dimensional modeling provides great support for this type of expansion, as concluded in the literature review.

With the help of conformed dimensions, new fact tables can be added and they can be linked to existing and new dimensions without affecting the already existing data. It is also easy to further enrich the incoming data by adding new attributes to both the fact tables and the dimension tables. The data warehouse can then be used for various types of analytics, e.g.

- Basic BI and reports, such as revenues and forecasting
- Customer segmentation
- Churn analysis
- OLAP-cubes

5.5.5 DEVELOPMENT AND IMPLEMENTATION

For the actual development, SQL Server 2017 Developer version was chosen as the database environment, as the developer version is free and allows development in a non-production environment with full features of SQL Server. SQL Server Machine Learning Services is a great addition, which allows the deployment of the R script inside SQL Server in a stored procedure, resulting in a very clean and compact solution.

The star schema was developed in SSMS using "CREATE TABLE"-statements. All needed FK- and PK-constraints were added to the fact and dimension tables. A separate schema was created for the staging area to separate the staging area from the actual data warehouse.

To test actual functionality of the system, some data was used. The data used is sales data of an online retail business, called "Online Retail Data Set", and can be acquired from the UCI Machine Learning Repository (2015). As source data and systems differ widely in companies and the actual ETL-process from the source to the star schema is out of scope of this thesis, the data was loaded by manual queries into the star schema by first importing the data into a staging table that mimics the structure of the xlsx-file. The source file has the following columns and meanings:

- InvoiceNo: The invoice number
- StockCode: The product code, contains both numbers and letters
- Description: A short textual description of the product
- Quantity: The quantity of the product sold
- InvoiceDate: Datetime format: 01/12/2010 08:26
- UnitPrice: Price of the product sold
- CustomerID: Customer identifier
- Country: Customer country

All data was loaded into the staging table. First, the product dimension was loaded by finding all distinct StockCode-values, and by finding the maximum value of the descriptions by using a “GROUP BY”-clause to eliminate duplicate StockCode-, and Description values. As the real values of the customer dimension are not important for the testing of the architecture, the country value was aggregated with a “GROUP BY”-clause in a way that the maximum value was picked and all rows where set to be valid with a Valid_From-value of ‘20091231’.

The date dimension was populated using example scripts available on the web. Finally the fact table was loaded by joining the staging table to the dimension tables on their corresponding columns to find the actual values for the key-columns. Besides the key-columns, the values of the staging-columns InvoiceNo, UnitPrice/Quantity and Quantity was loaded into the columns Sales_No, Amount and Quantity.

The execution of the market basket analysis was performed by embedding the R-script into a stored procedure in SQL Server. The prerequisites for this was to install SQL Server Machine Learning Services and to install the required packages for market basket analysis in R. The stored procedure takes the input dataset from the view of the dimensional model, with a “WHERE”-clause that controls the date. To also test the usability of the results, the script plots the top rules that are mined and saves the plot to disk.

5.5.6 RESULTS

The developed star schema is very easy to understand. The querying of the model is straightforward, and creating a view to use as an input for the MBA keeps the structure simple and understandable. The execution of the R script inside a stored procedure works, and it produces a plot of top rules and saves the plot as a .jpeg-file on disk. The execution of the stored procedure is however slower compared to running MBA as standalone in R with the same data. With the full sample dataset, the run time of the procedure with 0.01 support was 4 min 24 sec compared to roughly 2 seconds when running the R script independently. However, the returned messages of the script in the stored procedure show that generating the rules takes less than a second which would indicate that the bottleneck is caused by something else. By reducing the amount of data, the execution time is significantly reduced.

As the data is easy to query from the dimensional model, it would be easy to build new solutions on top of the model. It is also quite straightforward to add new columns, and adding new fact- and dimension tables would allow for easy expanding of the model. This means that the model seems to work well for other types of analytics. No automation was implemented, but it would be easy to implement by creating a SQL Server agent job that triggers the needed stored procedures and schedule it to execute in wanted intervals. The overall results of the implementation seems to be that the architecture and model works as expected.

6 CONCLUSION

To answer the research questions, an artifact was created. This artifact consisted of a 2-layered data warehouse and an analytics layer with market basket analysis implemented in R. A data warehouse provides a great platform for data storage and analytics. Dimensional models are easy to understand and query, and they provide good query performance. By using a two-layered data warehouse, the development is also quick and easy. The data warehouse functions as a great foundation for market basket analysis, and allows for easy expandability to support more data and analytics.

The creation of a simple dimensional model is straightforward. The most time-consuming part of the development is the development of the ETL-process and the actual design of the data warehouse. There are also multiple choices to be made when developing a data warehouse regarding database engines, ETL-tools and technologies, but the design itself is quite independent of these factors.

The Apriori-algorithm is fast, as it generates rules in a few seconds with the sample dataset. Generally speaking, the rules were mined in less than one second on the sample data. This meant that there was no point in trying other algorithms, as the speed of Apriori was more than enough. With different data, the situation might however be different, and could then require further experimenting with other algorithms and technologies.

Implementation of Apriori in R is simple, as association rules can be generated with only a few lines of code. It will however require more effort to find the correct parameters and to do actual data exploring. Embedding the R script into a stored procedure is however not as easy as running the R script as a standalone solution. Apriori requires the data to be in transaction format, and the function that transforms it into transaction format requires the source data to be in a .csv-file. When embedding the script in a stored procedure the goal is to use the data directly from the database which means that the function in question cannot be used. The data can however be converted to a transaction object by manually transforming the data to a matrix, but this requires additional code.

The R script was significantly slower when embedded in a stored procedure. As the message from R was that the rule generation was completed in less than a second, it is more likely that something else is slowing the script down. The actual bottleneck seems to be caused by the need to format the data into transaction format, and not by the Apriori algorithm. It will usually also take a few seconds for SQL Server to start a new R process when executing the stored procedure. However, the obvious benefit of the embedded script is that analytics can be kept close to the data and the costs and security risks associated with data movement can be eliminated.

As the development of the actual R script started as a stand-alone script unrelated to the data warehouse, and as the script works in-database, it is obvious that the architecture would also work if the script was not inside a stored procedure. An architecture with a separate R script would be relevant for older versions of SQL Server and other database engines. If the performance is too slow for the in-database solution, it would also be better to run the script as stand-alone and import the data from a .csv-file.

There are however some challenges related to the system:

Interval of script execution: It might take some trial and error to figure out an optimal interval for running the MBA-script. Running the script too often will not provide any new value as the rules will not change much, or there is not enough data to mine relevant rules.

Amount of data: It might be hard to figure out how old data should be used when running the script. As an example, should the script be executed once a month with only sales data of the past month?

Useful rules: When automating the process, it can be quite hard to maintain quality and usefulness in the generated rules.

Data cleansing: There might be existing and new errors in the data, which can cause problems in the ETL-process or the MBA-script.

Product dimension design: The product dimension can be modeled in many ways. For example, if a label for a product is changed, should the historical facts also be linked to the new product label, or should the dimension store the old value as well?

It is also questionable if an automated market basket system will be able to deliver relevant results, as it might be necessary to configure the script constantly. Further analysis might be required each time that MBA is performed to actually get relevant results. If the company only needs to do MBA occasionally or on an ad-hoc basis with further analysis, it is most likely an easier and better option to just obtain the data as needed and perform the analysis manually. However, the data warehouse with the star schema will provide a robust and easy-to-query platform, and works as an excellent base for both automated and ad-hoc analytics.

The implemented solution was not automated, but the actual automation of the system is fairly easy with different scheduling options discussed in the thesis. The system could also be further developed to be more reusable in different organizations by adding more variables and making the script more dynamic. By writing proper SQL-commands, it would be possible to create a "master"-script that would implement the entire system. This would allow for easy reusability and great flexibility with the addition of more variables. The only thing that would be left to develop is the actual ETL-process from source systems to the star schema, which is different for organizations. By developing the system further, it would start to resemble a generic product.

With the artifact, it is possible to answer the research questions:

1. *How should a data warehouse be designed so that it can support market basket analysis?*

There are various ways to develop a data warehouse. The proposed 2-layer architecture designed according to dimensional modeling is a fairly simple structure to implement, and serves the basic needs for market basket analysis perfectly. This architecture is fast to implement and it allows for easy expansion. However, if the business has higher requirements for data quality and auditability, a 3-layer architecture might be needed.

2. *How should automated market basket analysis be applied?*

The proposed solution in this thesis shows one way of implementing an automated MBA system. The architecture in itself is fairly technology-independent, as if SQL Server Machine Learning Services is not available, the R script can be triggered separately. This architecture can also be migrated to the cloud as-is when using a virtual machine. The architecture is easily automated in various ways, and by increasing the amount of variables the system will be more flexible. Market Basket Analysis can however be applied in many other ways as well with different software or programming languages.

6.1 FUTURE RESEARCH

How well does the star schema support expansion to more data and different types of analytics?

Further research should focus on investigating how well a data warehouse based on dimensional design can handle expansion, and how non-relational data can be incorporated to the model. It is also of interest whether a dimensional design will provide the best foundation for an ever increasing variety of analytics, as some other type of solution might prove to work better, especially when the volume and variety of the data increases.

How effective is an automated system compared to ad-hoc analytics?

It is questionable how useful an automated market basket analysis system will be in practise. This is most likely highly dependent on the business and the data. The results might prove to not be useful, and incorporating these results might even lead to poor decisions and campaigns. This means that in many cases, simple ad-hoc analysis might prove to be more cost-effective with greater results.

How effective is automated use of the MBA-results?

The results of an automated MBA-system can be used for many purposes. It is also possible to build sophisticated applications and further analytics that use the results. Some examples are recommender systems, marketing emails and group pricing. Further usage options and their effectiveness should thereby be researched.

How should MBA-results be visualized?

The visualization of the results are out of scope for this thesis, which means that this is a great area to work further on. In the implementation, the results were printed out as a simple graph, but there are much more sophisticated visualization methods. This means that different ways of presenting association rules could be researched further.

Can the proposed system be developed into a product?

The developed system shows some hope for the development of a generalized product that could be used as-is in different businesses. Improving the R script by adding more dynamic parts and using variables could make the system highly flexible. This means that the entire system could be deployed in seconds with a single build script. The remaining work would focus on the ETL from the source systems to the star schema.

7 SUMMARY IN SWEDISH

Utveckling av ett automatiserat system för korganalys inom detaljhandel

Företag samlar en stor mängd transaktionsdata som de skulle kunna dra nytta av men eftersom transaktionsdata ofta är sparat i operationella system är det orimligt att utveckla analytiska lösningar som utnyttjar dessa operationella data direkt. En lösning till detta är att utveckla ett datalager (eng. *data warehouse*).

Ett vanligt sätt att dra nytta av transaktionsdata är att göra korganalys (eng. *market basket analysis).* Med hjälp av korganalys är det möjligt att identifiera vilka produkter som ofta köps tillsammans. Den här informationen kan sedan utnyttjas för t.ex. produktplacering, -kampanjer och -rekommendationer. På basen av ett datalager är det möjligt att utveckla ett automatiserat system för korganalys.

Enligt egen erfarenhet, finns det inom industrin ofta bristfällig kunskap om datalagermodellering och analytik, speciellt vad gäller olika metoders för- och nackdelar. Målet med den här avhandlingen är att studera olika alternativ för datalagerdesign och att redogöra för olika algoritmer som används för korganalys. Det är sedan meningen att planera och utveckla ett system som består av ett datalager och korganalysintegrering. Systemet kommer slutligen att implementeras för att testa systemets funktionalitet och analysera resultaten.

Enligt Inmon m.fl. (2008), är ett datalager en samling av subjektorienterade, integrerade, tidsberoende och konstanta data som kan stöda ledningen i beslutsfattningsprocesser. Dessa sparade data är ofta av olika typ och genereras av olika affärsaktiviteter (Bojičić et al. 2016). I enlighet med Linstedt & Olschimke (2016), kan ett datalager planeras i två eller tre olika skikt. En tvåskiktsstruktur består av ett iscensättningsskikt (eng. *staging-layer*) och av själva datalagret. Eftersom målet med iscensättningsskiktet är att spara källdata i ursprunglig form liknar skiktets struktur strukturen för källdata. Iscensättningsskiktet minskar belastningen på operationella systemen. Treskiktsstrukturen har ett mellanskikt som är modellerat enligt 3NF. Det här skiktet lagrar rådata och påminner mer om ett operationellt system. Ovanpå det här normaliserade skiktet finns ett eller flera datatorg (eng. *data mart*) som ofta är baserade på dimensionell modellering. I den tvåskiktade strukturen är å andra sidan själva datalagret utvecklat enligt dimensionell modellering.

Enligt Kimball & Ross (2013) och Linstedt & Olschimke (2016), är dimensionell modellering väl accepterat i industrin och en standard datalagermodelleringsteknik. De två främsta fördelarna med dimensionell modellering är att den kan framföra data på ett lättförståeligt sätt samt ger en snabb förfrågningsprestanda (eng. *query perfomance*). Dimensionella modeller skiljer sig från transaktionella databaser som oftast är i 3NF. Databaser i 3NF strävar efter att minska redundans, vilket i sin tur leder till ett stort antal tabeller. Tabellnätverket som 3NF ger upphov till är effektivt för transaktionella system men svårt att navigera som användare.

Dimensionella modeller implementeras som stjärnscheman (eng. *star schema)*, vilket illustreras i Figure 5. Stjärnscheman består av två slags tabeller: fakta- och dimensionstabeller. I faktatabeller sparas data från affärsprocesser. Data från en affärsprocess skall sparas i endast en faktatabell. Varje rad i en faktatabell bör vara lika detaljerad, d.v.s. en rad i en faktatabell kan t.ex. innehålla information om endast en produkt. I Figure 6 illustreras en faktatabell (Kimball & Ross, 2013).

I Figure 7 presenteras en dimensionstabell. Dimensionstabellerna är avgörande för den dimensionella modellen. En dimensionstabell innehåller all ytterlig information till faktatabellen. Dimensionerna svarar på frågorna vem, vad, var, när, hur och varför till händelserna i faktatabellen. Varje dimensionstabell har en primärnyckel (eng. *primary key*), vilken används för att länka faktatabellen till dimensionstabellen. Genom att kombinera fakta- och dimensionstabellerna får vi ett stjärnschema (Kimball & Ross, 2013).

Det finns även andra metoder för att modellera datalager. I enlighet med Inmons alternativ modelleras själva datalagret i 3NF. Därefter utvecklas flera datatorg som fyller olika krav som har framställts av företaget och dess avdelningar (Inmon m.fl., 2001). Ett datavalv (eng. *data vault*) å sin sida byggs upp av knutpunkter, länkar och satelliter (eng. *hubs, links and satellites*). Knutpunkternas uppgift är att spara affärsnycklar (eng. *business key*) tillsammans med metadata. Eftersom affärsobjekt sällan förekommer ensamma behövs länkar som binder ihop knutpunkterna med varandra. För att kunna spara alla data räcker knutpunkterna och länkarna inte till. Satelliter lagrar all annan information som inte ingår i vare sig knutpunkter eller länkar (Linstedt & Olschimke, 2016).

Ett datalager fungerar som bas för t.ex. korganalys. Det finns flera olika algoritmer som kan utnyttjas för korganalys. En typisk algoritm för korganalys är Apriori-algoritmen (Yabing 2013, Rathod et al. 2014). Andra kända algoritmer är t.ex. Eclat och FP-Growth. Med hjälp av korganalys kan företaget bl.a. lägga till nya produkter, förändra sortimentet, korsmarknadsföra produkter, förändra produktplaceringen eller skicka skräddarsydda e-postmeddelanden i marknadsföringssyfte (Jabeen, 2018).

I den här avhandlingen strävar jag efter att hitta en lösning på hur man kan designa datalager som stöder korganalys. I enlighet med min litteraturöversikt, föreslår jag en tvåskiktsstruktur med korganalysintegrering enligt Figure 29. Själva datalagret bör vara modellerat enligt dimensionell modellering och korganalysen i analysskiktet bör utföras med hjälp av R. Systemet utvecklas med SQL Server som grund och korganalysen utförs med Apriori-algoritmen. Apriori finns implementerad i R-paketet "arules" och korganalys kan göras med endast några få rader kod.

Det var enkelt att implementera systemet. Testdata användes för att ladda datalagret och för att utföra korganalys. Systemet fungerade bra och skulle ha varit enkelt att automatisera. Själva korganalysen tog längre tid då R-skriptet var inbäddat i en lagrad procedur i SQL Server. Ifall R-skriptet är för långsamt med det dataset som används skulle det möjligen vara bättre att köra R-skriptet skilt från SQL Server. Det förblir dock oklart om ett automatiserat korganalyssystem klarar av att ge relevanta resultat, det kunde istället vara bättre att göra manuella analyser.

REFERENCES

1Keydata. Website, URL:
https://www.1keydata.com/datawarehousing/slowly-changing-dimensions-type-3.html
Accessed 24.2.2019

Abramson, I. Data Warehouse: The Choice of Inmon versus Kimball, IAS Inc.
URL: https://www.ismll.uni-hildesheim.de/lehre/bi-10s/script/Inmon-vs-Kimball.pdf
Accessed 14.2.2019.

Agile Alliance. Minimum Viable Product (MVP). URL:
https://www.agilealliance.org/glossary/mvp/#q=~(infinite~false~filters~(tags~(~'mvp))
~searchTerm~'~sort~false~sortDirection~'asc~page~1) Accessed 22.2.2019.

Agrawal, R. & Srikant, R. (1994) Fast Algorithms for Mining Association Rules. VLDB '94 Proceedings of the 20th International Conference on Very Large Data Bases. 487-499.

Aljabre, A. (2012) Cloud Computing for Increased Business Value. International Journal of Business and Social Science. 3(1). 234-239.

Baer, D. & Chakraborty G. (2013) Product Affinity Segmentation Using the Doughnut Clustering Approach. SAS Global Forum.

Bojičić I., Marjanović Z., Turajlić N., Petrović M., Vučković M. (2016) A Comparative Analysis of Data Warehouse Data Models. 6th International Conference on Computers Communications and Control (ICCCC). 151-154.

Breslin M. (2004) Data Warehousing Battle of the Giants: Comparing the Basics of the Kimball and Inmon Models. Business Intelligence Journal

Carroll, M., van der Merwe, A., Kotzé, P. (2011) Secure cloud computing: Benefits, risks and controls. 2011 Information Security for South Africa. 1-9. IEEE

Chegg, Website. URL:
https://www.chegg.com/homework-help/questions-and-answers/er-diagram-show-operational-system-retail-sales-business--create-complete-information-pack-q20412993 Accessed 18.3.2019

Disoln, Website. URL:
http://www.disoln.org/2013/04/SCD-Type-4-a-solution-for-Rapidly-Changing-Dimension.html Accessed 24.2.2019

Gartner (2018) Best Data Integration Tools Software of 2018 as Reviewed by Customers. URL:
https://www.gartner.com/reviews/customers-choice/data-integration-tools
Accessed 18.3.2019

Gulalkari, N. (2016) Implementing Apriori Algorithm in R. URL:
https://www.r-bloggers.com/implementing-apriori-algorithm-in-r/ Accessed 21.2.2019.

Hahsler, M., Buchta, C., Gruen, B., Hornik, K., Johnson, I., Borgelt, C. (2019) Package 'arules'. R Documentation. URL:
https://cran.r-project.org/web/packages/arules/arules.pdf Accessed 9.3.2019.

Hevner, A., March, S., Park, J., Ram, S. (2004) Design Science in Information Systems Research. MIS Quarterly, 2004, 28(1), 75-106. URL:
http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.103.1725&rep=rep1&type=pdf Accessed 16.3.2019

Inmon, W., Strauss, D., Neuschloss, G. (2008) DW 2.0: The Architecture for the Next Generation of Data Warehousing, Elsevier.

Inmon, W., Imhoff, C., Sousa, R. (2001) Corporate Information Factory. Second Edition. John Wiley & Sons.

Jabeen, H. (2018) Market Basket Analysis using R. DataCamp. URL:
https://www.datacamp.com/community/tutorials/market-basket-analysis-r#firsthead
Accessed 6.3.2019

Jukic, N. (2006) Modeling strategies and alternatives for data warehousing projects. Communications of the ACM, 49(4), 83-88.

Järvinen, P. (2004) On Research Methods. Opinpajan kirja.

Karthiyayini, R., Balasubramanian, R. (2016) Affinity Analysis and Association Rule Mining using Apriori Algorithm in Market Basket Analysis. International Journal of Advanced Research in Computer Science and Software Engineering, 6(10), 241-246.

Kimball, R. (1998) Surrogate Keys. Kimball Group. Website, URL: https://www.kimballgroup.com/1998/05/surrogate-keys/ Accessed 24.2.2019

Kimball Group. Dimension Surrogate Keys. Website, URL: https://www.kimballgroup.com/data-warehouse-business-intelligence-resources/kimball-techniques/dimensional-modeling-techniques/dimension-surrogate-key/
Accessed 24.2.2019

Kimball, R. & Ross, M. (2013) The Data Warehouse Toolkit: The Definitive Guide to Dimensional Modeling. Wiley.

Kumbhare T.A. & Chobe S.V. (2014) An Overview of Association Rule Mining Algorithms. International Journal of Computer Science and Information Technologies, Vol. 5 (1), 2014, 927-930.

Larose, D.T. (2005) Discovering Knowledge in Data. Wiley

Li, N., Zeng, L., He, Q., Shi, Z. (2012) Parallel Implementation of Apriori Algorithm Based on MapReduce. 13th ACIS International Conference on Software Engineering.

Linstedt, D. (2010) Data Vault Versus Dimensional – Part 1.URL: https://danlinstedt.com/allposts/datavaultcat/data-vault-versus-dimensional-part-1/ Accessed 12.2.2019

Linstedt, D. & Olschimke, M. (2016) Building a Scalable Data Warehouse with Data Vault 2.0. Elsevier.

Microsoft (2019) Available SQL Server 2017 editions. URL:
https://www.microsoft.com/en-us/sql-server/sql-server-2017-editions
Accessed 23.2.2019

Han, J., Pei, J., Kamber, M. (2014) Data Mining: Concepts and Techniques. Elsevier Science

Rathod, A., Dhabariya, A., Thacker, C. (2014) A Survey on Association Rule Mining for Market Basket Analysis and Apriori Algorithm. International Journal of Research in Advent Technology,2(3). 230-234.

Rittinghouse, J. & Ransome, J. (2009) Cloud Computing: Implementation, Management, and Security. CRC Press, Boca Raton

Singh, J., Ram, H., Sodhi,J.S. (2013) Improving Efficiency of Apriori Algorithm Using Transaction Reduction. International Journal of Scientific and Research Publications, 3(1).

Tan, P-N., Steinbach, M., Kumar, V. (2008) Introduction to Data Mining.

Tan, S.C, Lau, J.P.S (2013) Time Series Clustering: A Superior Alternative for Market Basket Analysis. Conference Paper: The First International Conference on Advanced Data and Information Engineering.

UCI Machine Learning Repository (2015) Online Retail Data Set. URL:
https://archive.ics.uci.edu/ml/datasets/online+retail Accessed 9.4.2019.

Videla-Cavieres, I.F., Ríos, S.A. (2014) Extending market basket analysis with graph mining techniques: A real case. Expert Systems with Applications,41(4), 1928-1936.

Yabing, J. (2013) Research of an Improved Apriori Algorithm in Data Mining Association Rules. International Journal of Computer and Communication Engineering, 2(1).

Zaki, M., Meira, W. (2013) Data Mining and Analysis: Fundamental Concepts and Algorithms. URL:

https://repo.palkeo.com/algo/information-
retrieval/Data%20mining%20and%20analysis.pdf Accessed 8.3.2019