Dimensional modeling – architecture and methodology

by Joakim Dalby (danish Dimensionel modellering, Datamodellering, Datavarehus)
I find knowing the theory really helps the practice.

1. Introduction and terminology

»The most valuable commodity I know of is information« said Gordon Gekko in Wall Street movie from 1987. »Knowing is good, but knowing everything is better« said CEO in The Circle movie from 2017. »Information at your fingertips« was a concept by Bill Gates in 1990.

Information has become a major asset for any organization. The book Das Kapital by Karl Marx could today get the title Die Information.

Purpose: Let's enable business users to make better decisions faster.

A data warehouse is the way to provide information to the business users and add value to the business and reuse data in a new way to increase the revenue.

A data warehouse (DWH) is a separate system, where a business user query and analysis will not slow down and not reduce the workload on an operational system.

A data warehouse contains a data model that is much simpler compared to a data model in a operational system where it can be complex to query normalized data. A data model in a data warehouse will organize the data for the purpose of analysis to the business users.

A dimensional model is a data model that puts a number into context by having a measure in a fact and a description in a dimension to create a KPI Key Performance Indicator value for the business in a fast and efficient way.

A dimensional model is a business-driven and user-friendly data model for analytical purposes by separating data into fact tables and dimension tables in a data mart as part of a data warehouse.

A data warehouse contains data from several operational systems, legacy systems or source systems and most important is it, that their data will be integrated, because integration or integrating of data is the keyword to represent an enterprise data platform with data assets to support data-driven decision-making processes within an organization. (In danish datadrevet beslutningstagning).
Make the right decisions at the right times.
A data warehouse solution must improve the decision-making capability!
Ask a business user what decision a dashboard or a report helps you to make?
What decision does this enable?

»Alot of you are going to have to make decisions above your level. Make the best decision that you can with the information that’s available to you at the time, and, above all, do the right thing« said Atul Gawande.

When an operational system forgets, a data warehouse remembers everything.

Let data storytelling lead to data products for business users before you think about data model and tools, architecture and methodology.

Illustration of Data warehouse models, read in sections 1.10.1, 1.10.3, 1.10.4

Illustration of Data warehouse architectures and data areas, read in section 1.11

Illustration of Dimensional modeling methodology, read in chapter 5

Illustration of the differences between OnLine Transaction Processing (OLTP) and OLAP OnLine Analytical Processing (OLAP) for an Order

From database design to Star schema and From business process to Star schema

Illustration of Star schema from Polaris data mart in Microsoft SQL Server database

Star schema with examples of dimensions and fact with data values

Star schema with examples of dimensions, bridge and fact with data values

Illustration of Star schema, Constellation schema, Snowflake schema

Illustration of fact table rows refer to slowly changing dimension type 2 table rows

Illustration of ETL Extract Transform Load

Description of Star schema with examples of dimensions and fact with data values

Data is stored in a data warehouse through an extract, transform and load process (ETL) where data is extracted from operational systems, transformed into high-quality data and checked for compliance with business rules and enriched by business users data before loaded into a data warehouse system.

Data quality isn't everything. It’s the only thing.

Gartner formulates that the platform for data and analytics is the heart of the digital business. Here you will find data management and the company's governance for data and analysis. Data is the raw material in our digital age. Data is collected, cleaned, controlled, optimized, shared and used, and our ability to work flexibly and intelligently at this »assembly line« is growing fast. Data management is crucial to tomorrow's business success. They must shape the technology and the data resource both short-term and long-term for the sake of both the existing business and the future's business. Now, the actual data platform is, with good reason, a focus area for the companies. In a data-centric world, tools, processes and competencies to handle data is infinitely crucial. All of today's greatest technological trends ranging from Internet of Things (IoT), Artificial Intelligence (AI), Virtual Reality (VR) and robotics are ultimately dependent on data managers, data architects, data modelers, data engineers, data stewards and company data strategies.

»That's all data is. A gift from yesterday that you receive today to make tomorrow better.« Jon Acuff: Finish: Give Yourself the Gift of Done, page 126.
»Description is not analysis.« Dr. Donald J. Wheeler: Making sense of data.

Performance Management Capabilities required:
- Planning – What do I want to happen?
- Forecasting – What will happen?
- Reporting – What happened?
- Dasboard – What is happening?
- Analytics – Why did it happen?

A data warehouse tells you what has happened (lag measures) - not why it has happened or what is going to happen (lead measures).

The four levels of analytics to get better data skills and data literacy are:
- Descriptive analytics use dashboards, scorecards and reports, e.g. reporting what happened in the past or is currently happening (facts). Help a business make more money.
- Diagnostic analytics is finding why things are happening or why it happened that way (insights). Help a business stop an inefficient process and save money.
- Predictive analytics does predict what is likely to happen in the future, a future result with data by using machine learning, statistics and algorithms, not with a crystal ball. Help a business capitalize on future trends and make tons more money.
- Prescriptive analytics does prescribe what should be done with the data and analytics. How to optimize a process. It means that organizations will get answers that will help them know what business decisions should be made. Help a business full potential through data-driven decision-making.

Contents
Chapter 1 is a large chapter because it covers a number of exciting topics that are good to put together:

1.1. Data warehouse definition

A data warehouse is a subject-oriented, integrated, time-variant and non-volatile collection of data in support of management's decision-making process. There are also plenty of operational decisions that are supported by a data warehouse.

The first four points are by Bill Inmon's definition with Dalby's explanation and the rest of the points are by Dalby.

- Subject-oriented because data from the business is merged into areas of subjects and not organized around the functional applications of the business.
- Integrated because data from multiple various operational systems, legacy systems or source systems is combined, merged and reconciled to provide consistent, common and uniform business names, definitions, formats, data types and units for get an aligned and a unified view. Integration or integrating of source data into a required shape for business analysis.
- Non-volatile because data do not change or historical data will never be altered, because it preserves historical data. Non-volatility implies that data in a data warehouse is read-only and cannot be auditable, overwritten, modified, altered, changed, updated, removed or deleted to ensure that source data is still accessible over time for enabling reliable historical analysis to analyze what has occurred, which is a purpose of a data warehouse.
- Time-variant because historical data is kept as current data at any given time. Time variance records change over time where the items of data is timestamped to inform when the data was registered. In contrast, an operational system will often only contain current data that is correct at the moment.
- Unite, interconnect, compile, conform and consolidate data into common format.
- Enrichment and processing of data for providing new knowledge and wisdom.
- Collection of data in support of management's decision-making process and analyzing the business with business performance measurement qualifiers and KPI Key Performance Indicator used as a measure of how a company is doing.
- Utilizing dimensional modeling for a dimensional model, a data model for supporting analyses where business users and analysts can easily understand and navigate the data structure and fully exploit the data for self-service BI.
- Goal to support slice and dice of data for business analysis and insight.
- Data-driven to become enjoyable and pleasurable to work with for the business users, and it is important for business and data analysts to have the right data, in the right form, at the right time, so they can turn data into insight and achieve business intelligence.
- Compliance requirement for provide proof that the reported numbers are accurate and complete data quality by having an audit trail for reconciliation check and auditability and traceability of the data for data lineage.
- Driven by business requirements specification or user story from stakeholders and business users. A data warehouse must have focus on the needs of the business.

An operational system often overwrites data and forgets data, whereby historical analysis to analyze what has occurred is not possible.

A data warehouse remembers everything by design to preserve historical data and to be non-volatile to data.

An example of subject-oriented is a sales process that is composed of several business process events as making an order, invoicing, payment, shipment, etc. Each of these business events may be handled by separate organizational units or departments or even by an external provider. Each of them would use various operational systems to do the job. A data warehouse modeling methodology is to fetch and integrate data into meaningful metrics for the business users.

Data from multiple various operational systems, legacy systems or source systems are stored in tables in a relational database called a data warehouse. Data will be merged and integrated inside a data warehouse to get a consistent format and use same naming.

Data warehouse modeling will do a decomposition of a business entity from an operational system and will make a new composition to another data model compared to the database design in the operational system to form a data warehouse according to the definition.

Dimensional model is a data model to fulfill Inmon's data warehouse definition. In our life we have three dimensions: Height, Width and Depth. Time becomes the fourth dimension as Einstein revealed. (In danish dimensioner Højde, Bredde,

Dybde og Tid). In a dimensional model, there are often several or many dimensions around some measures stored in a fact.

Time-variant data refers to data that changes over time. Time-variant data centers around its ability to monitor, store, and analyze data points across different periods. A unique time stamp is assigned to each data point, which allows tracking changes over time. Time-varying data is data that changes over time, such as stock prices, weather patterns or customer addresses.

Transversal data warehouse it cannot be said more tellingly.
(In danish tværgående datavarehus, mere sigende kan det ikke siges).

Enterprise data warehouse with integrated and consolidated data that contains a lot of a company's business data, e.g. about customers, sales, finance, employee, R&D data and many more data from several operational systems.
The term consolidate comes from the Latin consolidatus which means »to combine into one body.« To consolidate is to bring together and merge two or more items together.
(In danish virksomhedens datavarehus).

Data architecture describes how data is managed, from collection from several operational systems through to transformation, distribution and consumption. It sets the blueprint for data and the way that it flows through data storage systems.

Data integration is the essence of having a data warehouse to fulfill the purpose of integrated operational system data to business analysis by integrate data from multiple various operational systems, legacy systems or source systems to create a unified view that can be used for various purposes, such as analysis, reporting, or decision making and help business users to gain insights from their data, improve efficiency and reduce costs.

Data preparation is part of data processing as a process of preparing raw data from an operational system, legacy system or source system so data is proper for further data processing within a data warehouse.
(In danish dataforberedelse, datatilrettelæggelse).

Data cleansing and Scrubbing means that within a data warehouse, a process of identifying and correcting errors, inconsistencies, and anomalies in data and will often involve data validation, data transformation, data deduplication and removing duplicates and data reconciliation to improve accuracy and completeness of data, and to reduce the risk of data errors and biases. The result is cleansed data.
(In danish datarensning og skrubning, datavask for at højne data kvalitet).

Data uniformity means that within a data warehouse, a common data element from multiple source systems is referred to in a consistent manner.
(In danish ensartethed).

Data conforming means that within a data warehouse, a conformed column is created with a single and consistently labelled name that has the same common meaning with identical content across the entire data warehouse, e.g. same spelling of terms, same unit of measurement, same context descriptive data, text, hierarchy, group, band. The result is conformed data.

Hierarchy example for Global Industry Classification Standard (GICS) with levels Sector→Industry Group→Industry→Sub-Industry.

Important to use a specific unit of measurement (UOM) rather than different units as illustrated here: The winner of Miss World 1966, Reita Faria, had body measurements of 35-24-35 inches (89-61-89 centimeters) for bust, waist, hip with a height of 5 feet 8 inches (5′ 8″) (172.72 cm, 1.73 m) and a weight of 53½ kilograms or 117,95 pounds.

»Conformed dimensions are the glue that ties together your enterprise data warehouse«, said Kimball Design Tip #92.
(In danish overensstemmende, tværgående dimension, værdier, fakta, målinger).

Data enrichment and **calculation** means that within a data warehouse, a derived data is created, calculated or fetched a value from a variable into a **derived column** as an extra column or to replace a value of an existing column with new value in a data flow in a data pipeline into a table. Calculated measure is according to business rules to add useful data to the business users and display it in a BI tool. For example, a caller in a call center makes a registration of the begin-time and end-time, and a data warehouse in a fact table will calculate the call's duration to become a measurement, measure, metric, KPI Key Performance Indicator with unit minutes together with a CallDurationType dimension with a classification and band with intervals such as:

- less than 1 minute is Abnormal – Short.
- 1 to 2 minutes is Normal – Short.

- 2 to 5 minutes is Normal.
- 5 to 10 minutes is Normal – Long.
- more than 10 minutes is Abnormal – Long.

(In danish data berigelse, data beregning, data behandling, data bearbejdning, data sammenlægning, sammenstilling, samkøring).

Conforming - what to call it?
Humans all over the world are using many terms for the same thing. For example, a transport system: Subway in New York City, Tube or Underground in London, Metro in Paris or Copenhagen, U-Bahn in Berlin, Vancouver and Bangkok has SkyTrain as an elevated train system above the streets.
When a filipino wants to buy toothpaste in a store, he will ask for a colgate. A sign on the road says xerox instead of photocopying. In Denmark I will ask for a kleenex instead of a facial tissue. Some names of brands becoming words in our language.
It can be a little tuff job to find and to determine a good and common conformed term to become a column name, but later on the communication will be much easier among business people and IT-people.
How to present workload per day as 7½ hours, hh:mm 7:30 or decimal 7.50?
How to present a date, 2001-09-11 or 11-09-2001 or 9/11/2001?

For example IBM or I.B.M. or International Business Machines Corporation.

For example the length of a pipe is registered as 55 cm. in one source system and as 0.55 m. in another source system, but in a data warehouse the values become conformed to 550 mm.

For example, a sales revenue in different source systems in Australia, Canada, Singapore and USA is collected in terms of dollars: Australian dollars, Canadian dollars, Singapore dollars and United States dollars. In order to make sense of the revenues, the different types of dollars must be converted to a common base by factoring in the exchange rate. This factoring is a simple form of transformation to obtain comparable and conform amounts that can be summed up in one currency, e.g. US$.

For example, product data can be found in multiple source systems with different names of columns, different spelling of product names and different segmentations of products, therefore a data warehouse will unite, interconnect, compile, conform and consolidate the product data into one common product table with a conformed product name column to make it easy for the business users to make a list of all products of the enterprise.
(In danish konsolidere, samle sammen eller forene).

For example figures in DKK '000 means in DKK thousand, where a revenue 395,732 is 395,732,000 DKK.

For example, a status can be found in order statuses, shipping statuses and Payment statuses from multiple source systems where a data warehouse will unite and conform all statuses, maybe in a hierarchy or in a group.

It is important to design a data warehouse to support reporting and data analyses by a single and common data model that is easy to navigate in and a business user does not have to think of data across of source systems. Data integration becomes seamlessly. Many source systems update its data to reflect the most current and recent state while a data warehouse also maintain history.

For example, an ambassador who has lived in Washington, London and New Delhi for the last 25 years and always bought her perfume from a Lancôme store in Paris with a IT system that only stores current shipping address, all the sales over the last 25 years will be placed in India, while a data warehouse will remember all cities and give a correct picture of the sales shipped around the world.

Read about data processing in section 1.7.5. Data staging area.

Reference data or lookup data is data used to classify or categorize other data and are non-volatile, static or slowly changing over time. Reference data is like a list of countries with ISO country codes, Postal codes, currencies, colours, sizes, customer segments, Product type, Transaction type and Calendar data, and exists as lookup basic data in lookup tables.
Reference data is about the category or the context of a business entity and is concerned with classification and categorisation.

Master data is concerned with business entities such as customers, suppliers, employees, products, financial structures and is non-transactional data.

For example, adding a new customer or sales product is part of the standard business process, and adding a new product classification or a new customer type will result in a modification to the business processes to manage those items.

Systems for Reference Data Management (RDM) and Master Data Management (MDM) become source systems for a data warehouse solution.

1.2. Data mart and Dimensional modeling definition

Data mart with specific subject data from one or more business processes that contains data and information that is relevant for business function(s), department(s) and business users. A data mart has one purpose to customize and summarize data for tailored support of a specific analytical requirement from a business unit. It utilizes a common enterprise view of strategic data and provides business units more flexibility, control and responsibility. A mart display data used to support decision-making, reporting and dashboard (in danish udstiller). A data mart can be a data product.

For example, Sales mart, Customer mart, CRM mart, Churn prediction mart, Market mart, Procurement mart, Inventory mart, Production mart, Shipment mart, HR mart, Tax mart, Credit risk mart, Fraud detection mart, Financial mart, Budget mart, or a mart that combine data from CRM Customer Relationship Management, ERP Enterprise Resource Planning and PIM Product Information Management.

A data mart contains the data to calculate Key Performance Indicators (KPI) e.g. MRR Monthly Recurring Revenue and ESG Environmental, Social and Governance. (KPI in danish Nøgletal).

Dimensional modeling and the process of dimensionalization by thinking dimensionally is a methodology, an approach and a technique that seeks to divide and present data within a standard and intuitive framework of a dimensional model consisting of two elements, entities or tables with cleansed and conformed data.

- **Dimension** contains context descriptive data for textual, numeric, date, time, hierarchy, group, band to define and identify a fact.
  Is called descriptive data values, dimension values, dimension members, dimension properties or dimension data.

- **Fact** contains measurements, measures, metrics, KPI, analysis variables from a business process with data from a business process event for something that has happened, a transaction, a circumstance, an instance, an incident or a state.

A dimension and a fact have data values that are directly ready to be used by the business users.

A fact tells »how many«, »how much«, »how often«, »how long« or »how far« through measurements.

A dimension tells »who, whom, whose, what, which, where, when, why, how« through context descriptive data to measurements.

A general question for a business process: Who does what?

Example of question words and answers

- Who was involved? Who is? (People, Organization, Employee, Customer)
- What was involved? What is? (Product, Service, Experience, Treatment)
- What happened? What action was taking place? (Event, Transaction)
- Which thing happened? Which is? (Event, Transaction)
- Where did it happen? Where did it take place? (Location, Address)
- When did it happen? When did it take place? (Date, Time)
- Why did it happen? Why event occur? (Cause, Reason, Promotion, Weather)
- How did it happen? How did you get here? In what manner?
- How old are you? How tall are you? How hot is it?
- How many can be summed up to a total (Quantity, Amount)
- How many can be counted to a total (Number of times, DistinctCount)
- How much can't be summed up or counted (Price, Time, Temperature)
- How often asking about frequency (Shopping in the mall, Travelling by train)
- How long asking about a length of time (Duration, Taxi drive, Waiting time)
- How far asks about distance (From home to work, To Copenhagen, Running)

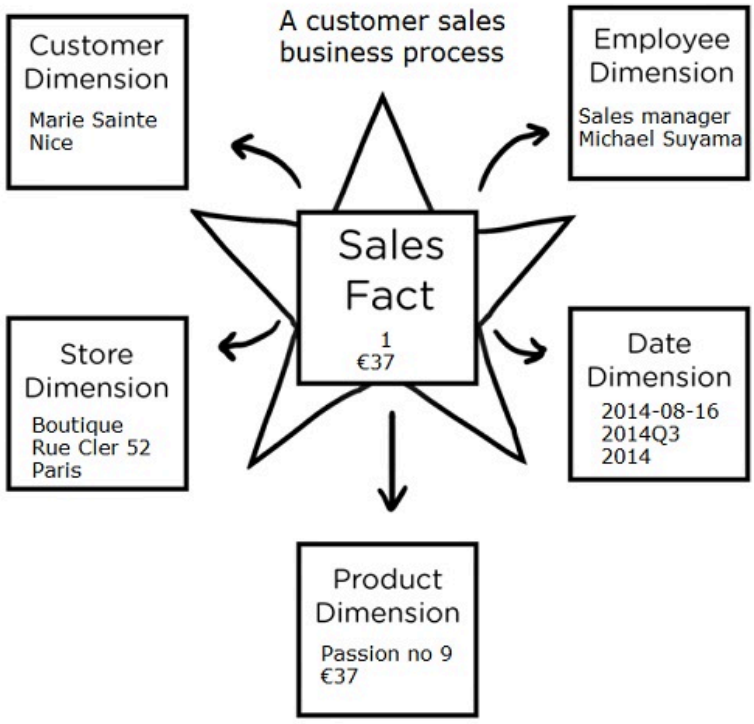
Example of a sales business process for customer purchasing of a product
A data storytelling about a customer purchase, and question words turn into dimensions and facts:
»Marie Sainte living in Nice purchases a Passion no 9 as a birthday present for €37 on August 16, 2014 in Boutique at Rue Cler 52 in Paris by sales manager Michael Suyama and she pay using her credit card.«

Question word	Data value	Dim/Fact
Who was buying	Marie Sainte	Customer
	Nice	Customer
What was bought	Passion no 9	Product

	€37 unit price	Product
Where did it happen	Boutique	Store
	Rue Cler 52, Paris	Store
When did it happen	2014-08-16	Date
Why did it happen	Birthday present	Fact
How was payment	Credit card	Fact
How many	1 quantity	Fact
How much	€37 sale amount	Fact
By whom	Sales manager	Employee
	Michael Suyama	Employee

A fact has multiple dimensions, which are displayed in a **star schema** diagram with dimensions surrounding a fact, or a fact surrounded by dimensions.

A star schema example with some of the data values from the above table, where measurements are placed in one fact and context descriptive data, text, hierarchy, group, band are placed in five dimensions:



A star schema diagram is implemented in a database as a fact table with dimension key columns connected to several dimension tables. Please click at the star schema diagram above to see some data values in database tables.

In the above example not all context descriptive data values become dimension values, because there is no Payment dimension with a payment method as Credit card, and there is no Reason dimension with Birthday present, therefore I let them remain in fact as tags for now.

(Fact in danish faktum, faktuelt, faktuelle, kendsgerning fra en forretningsproces, en hændelse, en begivenhed, en transaktion, en omstændighed, en instans).

A logical design has contextual entity and measurement entity.
A physical design has dimension table and fact table in a relational database that represents a data mart in a data warehouse using storage as RowStore or Column-store and data virtualization in BI tools.

A data warehouse can have multiple data marts bound together with conformed and shared dimensions by multiple facts.

A conformed and shared dimension is used across multiple facts where it supports integration of multiple source systems into a data warehouse/data mart and is essential for enterprise data warehousing.

Multiple source systems can handle a date and a time in different time zones, therefore it is a good idea to conform a source date or time to e.g. Universal Time Coordinated (UTC) or Central European Time (CET) to make it easier to compare

dates across source data and to bound it to a Date dimension and a Time dimension (calendar dimension and clock dimension).

A fact has conformed measures with same calculation methods and common unit of measure e.g. SalesRevenue and SupportRevenue in two measures are both pre-VAT (value added tax) in United States Dollar USD to make it easier to adding them for a result TotalRevenue.

Dimension and fact will focus on ease of business users (end users) accessibility and provides a high level of performance access to the data warehouse/data mart.

Dimensional modeling basic idea is to do a decomposition of a business entity often from a relational model and to make a new composition to a dimensional model where a business entity becomes dimensions and fact to form a data warehouse to preserve when data was happening, occurred, registered or took place in operational systems to form a data warehouse according to the definition.

Kimball page 37: »The BI team needs to understand the needs of the business as well as the realities of the underlying source data.« A business process-centric fact table supports integration of multiple source systems via a set of conformed dimensions, and the entire data warehouse is build process-by-process.

Kimball recommended to select the business process to model by gathering and combining an understanding of business needs from business requirements specification or user story and available data from the business process event with a business entity in an operational system or source system.

Ralph Kimball is the father of dimensional model with dimensions and facts through a dimensional modeling methodology and types of slowly changing dimensions in his book »The Data Warehouse Toolkit, The Definitive Guide to Dimensional Modeling, Third Edition« from 2013 and first edition was published in 1996.

I find dimensional modeling methodology for a dimensional model with dimensions and facts as a subset of Entity Relationship modeling methodology for a relational model with tables in a database. A Entity Relationship Diagram (ERD) for a database design can also be used to illustrate relationships between dimensions and facts in a star schema or a constellation schema, read more in section 1.10.4.
Kimball Design Tip #158 talks about a semantic layer as a mandatory component of a data warehouse architecture.

Peter Chen is the father of Entity Relationship modeling methodology and he said in his seminal paper in 1976: »The entity-relationship model adopts the more natural view that the real world consists of entities and relationships. It incorporates some of the important semantic information about the real world.« Read more.

Edgar F. Codd is the father of relational model and normalization methodology to normal forms to create a relational database with tables with primary keys and foreign keys in his paper »A Relational Model of Data for Large Shared Data Banks« from 1970.

Bill Inmon is the father of data warehouse back in the 1970's where he began to define the term data warehouse as a one part of the overall business intelligence system. A data warehouse is the foundation for business intelligence.
Inmon modeling is about that an enterprise has one data warehouse with entities in a relational model as a centralized repository, and multiple data marts which source their information from the data warehouse. In a data warehouse, data is stored in third normal form (3NF) relational model database.
A data mart is a simple form of a data warehouse that is focused on a single subject. Denormalization and data redundancy is a norm for a data modeling methodology in a data mart.

A relational model has no data redundancy.
A dimensional model has a large amount of redundant data.

Ralph Kimball said: »**A data warehouse is a union of all its data marts.**«

1.3. Data warehousing definition

Data warehousing is made up of three components The architecture, The data model and The methodology to help a data modeler, a data engineer and a data warehouse designer to build a comprehensive and reliable data warehouse system with data from several operational systems that will be integrated and keep historical data.

Relational model and Dimensional model are two approaches to data modeling that are often used in data warehouse design.

Inmon modeling is a top-down approach and technique for a data warehouse architecture and methodology using Entity Relationship modeling to a relational

model.

Dimensional modeling is a bottom-up approach and technique for a data warehouse architecture and methodology where a data mart is formed based on the business needs to optimize analysis and reporting into a dimensional model with dimensions and facts in a star schema.

Business requirements may change over time, which can impact a data model. This requires close collaboration between the business stakeholders and the data modeling team. Also, it is essential to ensure that a data model is flexible and can accommodate changes in business requirements specification or the user story.

Data can be complex, with multiple sources and formats. The challenge is creating a dimensional model to accommodate this complexity while ensuring the data is: accurate, consistent and complete. Read more about data quality in section 1.14.

Data consistency is crucial in dimensional modeling, ensuring the data is accurate and reliable. The challenge is maintaining data consistency across multiple data sources and ensuring that the dimensional model is updated as new data becomes available.

One of the biggest challenges in dimensional modeling is ensuring data quality. If the data is not accurate or complete, the analysis and reporting may be incorrect, leading to wrong business decisions. It is essential to ensure the data is clean, complete, and consistent before beginning the modeling process.

A OLAP (OnLine Analytical Processing) cube for Multidimensional model or Tabular model can be at the top of a dimensional model to present data in a data access BI tool like Excel, Power BI, Tableau, QlikView, Qlik Sense, Alteryx or Targit for reporting and dashboards.

Kimball's data warehousing architecture, known as Enterprise Data Warehouse BUS Matrix (Business User Support) is a collection of conformed and shared dimensions that has the same meaning to one or more business processes (facts), illustration, read more about dimensional modeling methodology in chapter 5.

Business Intelligence (BI) system provides the information that management needs to make good business decisions. BI is going from data and information to knowledge and wisdom for the business users. BI was previously called Decision Support System (DSS) (in danish Beslutningsstøttesystem, Ledelsesinformationssystem LIS). BI could also stands for Business Insight (in danish indsigt, indblik).

Data warehouse architecture, data area and data operation, see illustration
Data does not turn into insight without effort of building a good data warehouse with an architecture of data areas or data layers or data zones, a data model for each data area and a methodology for process of data in each data area with the famous acronym **ETL** that stands for three operations on data:

• Extract	• Extracting	• Extraction
• Transform	• Transforming	• Transformation
• Load	• Loading	• Loading

Extract is the process of fetching data and extracting a desired subset of data.
Transform is the process of integrating the extracted data into a required shape.
Load is the process of loading the transformed data into a suitable target table.

ETL process brings multiple source systems together and combines their data to an integrated set of data into a data warehouse and into data marts. Read about data warehouse architecture and ETL process in section 1.11 with explanations for the illustration.

In the article I will focus on the basic concepts, principles, terminology, architecture and methodology for Dimensional modeling. My homepage has other articles of how to design and implement a data warehouse system with levels of data in data areas or data layers through a ETL process and orchestration to orchestrate a set of tasks by a tool to automate data flows, managing and scheduling pipelines and workflows to ensure tasks are executed in the correct sequence and there is automated interoperability. ETL scheduling illustration.

1.4. Database definition

Let's have many database concepts as a common framework.

A data model represents correctness, completeness and consistency.

A database schema is a logical grouping of and a container for database objects e.g. tables, views, table-valued functions and stored procedures, and it can handle user access rights.

A table contains data in rows also known as records, tuples, instances or data rows.
A table contains columns also known as attributes, fields, elements or properties.
A data table thousands of years old

A column contains an atomic value like a date, a number, a amount, a name and a text like a ProductName and Description. A compound value like a Full name of a person, can be further divided into three columns for FirstName, MiddleName and LastName or Surname or Familyname. Sometimes FirstName is called Givenname.

Null is the absence of a value. No one knows what it is, because it is missing.
Sometimes called a blank value, an empty value, an indefinite value, a nothing or a none in a column which value is missing at the present time, because a value is not yet registered, not yet assigned, a value is not defined for a row and it is okay that a column has no value. Some data engineers say that null represents the unknown or the nonexistent.

A column can contain null (no value), or a column must contain a value = must not allow null, must have a constraint not null to enforcing it to always contain a value, maybe through a default value.

In a Customer table a column LatestPurchaseDate has a row with null that indicates the lack of a known value, or it is known that customer hasn't made a purchase yet, therefore it is known to be absent or omitted or missing.
For example in sql: WHERE LatestPurchaseDate IS NULL.

When a person column Date of Birth (DOB) is null I will say it is missing because of the person prefers not to say or a lack of the information at the time the person was registered – a not knowing situation.
A default value could be used like 0001-01-01, but the person's age is getting too old.

When a person column Date of Death (DOD) is null I will say that the person is alive and is living, but it could be better to use value »Present« when I know the person is alive. When a person has died and I don't know when, it could be better to use value »Died« until I have the date. Null could represent that I »Don't know«.
In a logical model there will be a sub-entity with column DateOfDeath not allow null and a row will represent a deceased person from a super-entity Citizen or Person.

When a person column MiddleName is null, I will say it is nothing because it is okay that there is no value because a value is not required and therefore is not missing – a knowing that there is nothing situation. Sometimes people for MiddleName typein a value »(None)« to tell that a person has no middle name, or »(Missing)« to tell that a person has a middle name but we don't know it, and let null indicates that we don't know if a person has a middle name or not or typein »(Unknown)«.
An alternative is to have another column MiddleNameStatus, but I think we in most cases accept a null in MiddleName for none or nothing, until we know the name if it exists and type it into the system.

A JobInfo column with values Banker, Lawyer, Teacher, Unemployed, Job unknown and not allow null.

A column Number of items is going to be assigned a value that is too big or too small to be represented for the data type, maybe because of an error, and therefore we are replacing the value with null and adding an audit description »Out-of-bounds value« to an audit column for later examination and investigation. On the other hand I would prefer column Number of items to be not allow null and a use of a negative value could be interpreted as a return of an item.

A numeric column with null in several rows can cause a problem, because queries using aggregate functions can provide misleading results. A SQL statement like this: Sum(<column>) can return one row that contains null, or use of Outer Join can result in a column with null.

In a division it is nice to use Null functions to avoid division by zero error.
For example in sql: SELECT ISNULL(SaleAmount / NULLIF(NumberOfItems, 0), 0).

Null reminds me of: »No answer is also an answer« when I ask a person a question and he or she does not reply, his or her very silence is an answer in itself.

When I receive a Christmas gift in a box with something inside, I do not know the contents of the box, therefore the gift is unknown to me, and known to the giver. I will not say that the gift is null for me, okay maybe null if the box is empty but I only know that, when I open the box and I could say the gift is missing.

Joe Celko's SQL for Smarties: Advanced SQL Programming, said: »Null is not a value; it is a marker that holds a place where a value might go.« Null is a place-holder for a future value, e.g. an Address when it is known from the customer.
For example in sql: WHERE CustomerAddress IS NULL.

Opposite term of Null can be Assigned, Extant, HasValue, IsSet, Usable, Something might imply not nothing or not null.
For example in sql: WHERE CustomerAddress IS NOT NULL.

The question is how you will interpret null when you are reading it?
An orphan child as a row in a Child table with a column MotherId as Null. Another orphan child has also MotherId as Null. It is good that Null = Null will evaluate to Null, because I don't know if the two children are siblings or not, therefore no answer.

Empty string or zero-length string has zero characters with a value as "" or '' or String.Empty, and null string has no value at all. An empty string could be used as a default value for a column that does not allow a null. Trim a string value is taking away the spaces at beginning and at ending of the value, and when the string only contains spaces the trim makes it as an empty string.
[Alt+0160 at numeric keyboard gives a Non-breaking space (No-Break Space) that can be replaced to an empty string e.g. Replace(<StringColumn>,CHAR(160),'').]
To treat null, blank, empty, nothing and empty string as equal to null in sql:
WHERE (NULLIF(TRIM(CustomerAddress),'') IS NULL).

Void I have seen on a paper fly ticket with stopovers to mark the empty lines that there is nothing here, to avoid me to be able to add an extra stopover.

Unknown represents a value, therefore it is not null, but the value is not known in a system because it does not exists in the system, or it is a mistyping, a typos or a misspelling and therefore is incorrect or wrong. Unknown is something, null is anything and thus nothing. (In danish: Ukendt er noget, null er hvad som helst og dermed intet).

For example, I receive an address with a zip code I don't know therefore the zip code is an unknown value for me and I need to validate it and make the zip code value known for the system by insert the zip code value in a table of zip codes and cities and countries.

The cause of an event, usually a bad event, is the thing that makes it happen. Example of causes are: accident, crash, delay, disaster, flood, sickness. If something is unknown to you, you have no knowledge of it. The cause of death is unknown.

John Doe (for men) and Jane Doe (for women) are used when the true name of a person is unknown or is being intentionally concealed or anonymized.

Data type for a column defines the kind of data that can be stored such as integer, tinyint, smallint, numeric, decimal, string, text, date, time, datetime, amount or money or decimal(19,4), boolean (bit) with two values True (1, Yes) and False (0, No).

Sometimes a source system use a GUID as data type, Globally Unique Identifier, as 32 character hexadecimal value of numbers and letters from A to F, typically displayed in five groups separated by hyphens, as data type uniqueidentifier else char(36) in alphabetical sort order or as a 16-byte value. For example:

F32E64C4-3C07-4006-B922-25184D28DE0F

1st. The first 8 characters (4 bytes)
2nd. The next 4 characters (2 bytes)
3rd. The following 4 characters (2 bytes) [timestamp in version 1 OSF GUID]
4th. The next 4 characters (2 bytes) [clock sequence]
5th. The final 12 characters (6 bytes) [NodeID (i.e. MAC address)]

SQL Server sorts uniqueidentifier based on byte groups (looking at byte groups right-to-left and left-to-right within a byte group) starting from the ending 5th byte group and moving towards the 1st byte group reversing the order on the 3rd byte group from right-left to left-right. Therefore 6D is before 90 in the 5th byte group:
ED1EA42C-DD70-46DA-A088-6DBB506245D6
6FB2BD5D-48F3-48A3-8D19-909BE17338BC
GUID values are not naturally sequential [assigned by newid()] therefore data fetch of latest guid value is impossible. Seldome a source system uses newsequentialid().
A UUID (Universally Unique Identifier) version 7 value at https://uuid7.com/ or https://uuidv7.org/ is time-sortable, e.g. Fri, 2025-02-07T14:34:08.000Z gets:
0194e0d5-1300-7628-8680-9b9bd1989a0e convert to timestamp.

Important tip by naming columns, do not use possessive nouns such as Recipient's date of birth, better to use CamelCase like RecipientBirthDate for a column name, and don't use values in column name like EmployeeDay or NightCode, better use a column name as EmployeeShiftTypeCode.
Unique column naming is recommended.

Abbreviations and acronyms are discouraged.
For example, column name City, a better name is Customer_Mailing_Address_City.

With conforming or conformance we avoid synonym columns with same content but different names, and we avoid homonyms columns with same name but different content. For example, Type is a general term, better give a column a name e.g.: PersonType, ProductType, CustomerType, TransactionType, etc.

Jack Reacher said: There are four types of people who join the military:
- family tradition
- patriots, eager to serve
- just need a job
- the kind who want a legal means of killing other people

Names of tables and columns must be business-friendly and must make sense for both the business users and the IT developers. It is possible to use a view to make better names. For example, an IT developer named a date column in a table as ReservationExpiryDate and in a view to a business user the column is renamed to Date_of_Reservation_Expiry. Another example where BankAccountBalanceAmount becomes Saldo and IncludeTipsFlag becomes Tips with Yes and No values instead of database values True/False or 1/0.

A derived column represents a value that is derivable from the value of a related column or set of columns and not necessarily in the same table. For example, the age of a person is derivable from the date of birth column and date of today, or a calculation of an amount or a glue column to bind rows together for fast search.

A single-valued column holds a single value for an entity occurrence.
A multi-valued column holds multiple values, for example for a MovieCategory: »Children, Comedy« or »Action, Crime, Thriller«.

A candidate key is a unique identifier for all rows in a table so each row contains a unique value and it can be a null in one row. Ensures that each row is unique. For example, column EmployeeId, Employee Social Security Number (SSN), Employee Login name, Employee Passport Number or Employee Car number plate (or Car license plate or Vehicle registration plate).
A candidate key can be a collection of columns, e.g. EmployeeId + CourseId + DateOfCourse.

A primary key is selected among the candidate keys and is a unique identifier for each row in a table and its value is unique for all rows in a table, so each row contains a unique value and it can't be a null (not allow null, non null). For example, column EmployeeId is the primary key in an Employee table because it is an artificial auto-generated unique sequence number or an identity column generated by the database system to uniquely identify each row in the Employee table. Hence, it's often called a surrogate key. A car number plate or car license plate or vehicle registration plate is a surrogate key to make all cars unique in the traffic, and it must not allow null because it is illegal to drive without one.

It works nice for a database system based on a single-node architecture with all the data resides on a single node, and the system can guarantee that the data is written and stored at one single location. In a distributed architecture the work is spread across multiple nodes where each node is responsible for handling smaller chunks of the data. Ensuring that each node in a distributed system assigns unique values, without any overlap or data integrity violations, would require complex synchronization between nodes, which can then cause performance issues. It will be an application layer that does a calculation of a next value for column EmployeeId by finding the max value and add 1 to it.

Alternate key or Secondary key is the key that has not been selected to be the primary key, but it is still a candidate key and it can be a null in one row.

A compound primary key is concatenated data e.g. a US phone number contains area code + exchange + local number in an EmployeePhonebook table.
ISBN International Standard Book Number is concatenated of Country code, Publisher code, Serial number, Check digit.

A composite primary key is composed of multiple columns which combination is used to uniquely identify each row in a table e.g. column EmployeeId and column CourseNumber in a Participant table. Useful when a single column isn't enough to uniquely identify a row. (Sometimes called a concatenated key).

An entity is a set of objects with the same properties and is often implemented in a relational database as a table or as several tables with relationships, and a table has columns and rows to store data, where a primary key is unique and for use in identification of a row.

A foreign key (one column or a collection of columns) in a table refers to the primary key in another table to establishing relationships or links between the two tables.

A relationship defines how the entities are related to each other with a meaningful association among entities in three ways called one-to-one, one-to-many or many-to-many displayed in a Entity Relationship Diagram (ERD).

A relationship is implemented by including a primary key in a related entity and call it a foreign key to make sure of the consistency in a database. When a database is consistent we can navigate with a join among tables where relationships become navigation paths.

Referential integrity is a logical dependency of a foreign key for a primary key and is implemented as a referential integrity constraint to get a stable reference.

Example of a many-to-many relationship is between two entities Employee and Course, where one employee can take many courses and where one course can be taken by many employees, may be modeled through the use of a third entity that provides a mapping between them with foreign keys back to Employee and Course plus extra data e.g. date of course, evaluation of the course, student grade.

A recursive key is a foreign key that refers to the primary key of its own table for a parent-child relationship.

Normalization is the process of structuring a database in several tables to remove redundancy or reduce data redundancy and improve data integrity, through steps called normal form: 1NF, 2NF, 3NF, BCNF, 4NF and 5NF.
Guide to normal forms and theoretical.

Meanwhile a data warehouse loves data redundancy, because it is not a human that update data, instead it is a ETL process that runs as a computer software program developed by a ETL developer or a data engineer.

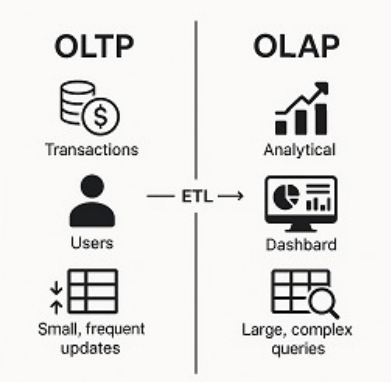
OLTP stands for OnLine Transaction Processing for a transaction-oriented database system such as mostly operational systems that process all kinds of queries on data as read, insert, update and delete. (On Line Transactional Processing).

OLAP stands for OnLine Analytical Processing for a analysis-oriented database system such as a data mart or a cube that process a query on data for optimized read for business users to do slicing for a a specific set of data and dicing for different viewpoints on data. Read more.

SQL stands for Structured Query Language e.g. Select From Join Where Union and is used in OLTP system and OLAP system.

OLAP system can have:
· MDX (Multidimensional Expressions) as query language for multidimensional cube
· DAX (Data Analysis Expressions) as query language for tabular cube

Data modeling methodology for OLTP system and OLAP system is very different and the use of the systems, but a ETL process binds them together.



A view is a stored sql query criteria statement used to fetch data from a database and the data set has columns as a table and is called a result set, rowset or a recordset. Also called a virtual table representing the result of a database query.

A materialized view is a database object that contains the result of a sql query e.g. data is being persisted into a table, a file or placed in a server memory cache. A view can be materialized through an index. Materialization is for performance reasons as a form of optimization of a ETL process and for fetching data to business

users e.g. five tables that are joined and aggregations are precomputed and precalculated. For a client-server database application, the result can be placed as a local copy at a users computer PC.

A table-valued function is an intelligent view with parameter as a mathematical function and it is returning a result set wrapped in a table to be used in a join.

A stored procedure (sometimes parameterized) performs actions on data and may return a result set, a value or an output parameter. Procedures are important for a ETL process.

Join is often between two tables columns of primary key and a foreign key. An equi join (equality join condition) includes matched rows in both tables by matching column values and is indicated with an equal operator =. A natural join is a type of equi join which occurs implicitly by comparing all the same names columns in both tables. The join result has only one column for each pair of equally named columns.

A theta join or non-equi join matches the column values using any operator other than the equal operator like <, >, <=, >=, !=, <>. A outer join includes both matched rows and unmatched rows of a table, normally the left side table else right or full from both tables. A self join when a table is joined with itself, sometimes through a sub-query with a Where clause, it is called nested sql statement.

SQL example:

```
SELECT COUNT(*) AS NumberOfBoats
FROM dbo.CopenhagenCanals
WHERE SailingPast = 'My window'
```

I prefer alias using = instead of AS:

```
SELECT NumberOfBoats = COUNT(*)
FROM dbo.CopenhagenCanals
WHERE SailingPast = 'My window'
```

Because assignment of a variable looks very much as my sql before:

```
DECLARE @NumberOfBoats int
SELECT @NumberOfBoats = COUNT(*)
FROM dbo.CopenhagenCanals
WHERE SailingPast = 'My window'
SELECT @NumberOfBoats
```

A relational database is characteristic by the acronym **ACID**:

- **A**tomicity where all parts and steps of a transaction commit succeed or if any step fails then the transaction are rolled back, rollback. Therefore a transaction cannot be left in a half done state. For example, in an application a transfer of funds from one account to another, the atomicity property ensures that, if a debit is made successfully from one account, the corresponding credit is made to the other account.
- **C**onsistency where all committed data must be consistent with all data rules like constraints, triggers, validations. A database will move from one consistent state to another consistent state when a transaction commit succeed or remain in the original consistent state when a transaction fails. For example, in an application a transfer of funds from one account to another, the consistency property ensures that the total value of funds in both the accounts is the same at the start and end of each transaction. It means a user should never see data changes in the mid transaction.
- **I**solation where every transaction should operate as if it is the only transaction in the database, where no transaction can disrupt or interfere with other concurrent transaction. Transactions that run concurrently appear to be serialized. For example, in an application a transfer of funds from one account to another, the isolation property ensures that another transaction sees the transferred funds in one account or the other, but not in both, nor in neither.
- **D**urability where once a transaction is committed and data persisted, the updated data rows are available for all other transactions, and data will survive system failures and can be recovered. For example, in an application a transfer of funds from one account to another, the durability property ensures that the changes made to each account will be recorded permanently and not be reversed.

This means that a transaction is either carried out completely successfully or not at all (Atomic), that only valid data is added to the database (Consistent), that transactions never affect each other (Isolated), and that transactions are never lost (Durable). Read more.

Lock granularity of database, table, extent, page or a row is the simplest way to meet the ACID requirement. A page is an eight-kilobyte storage area and an extent is a group of eight pages. Deadlock is when one resource is waiting on the action of a second resource while the second resource is waiting of the first resource, so

there is no way to finish and the database system will kill one transaction so the the other can complete.

1.5. Data, Information, Metadata definition

Data is the plural form of the Latin datum meaning »given, the given, certain«, and it is derived from the verb dare meaning »to give«.

Data is a formalized representation of facts, measurements, texts or ideas in such a form that it can be communicated or transformed by some process. Data is suitable for transmission, interpretation or processing by humans or automatic aids.

The data 38 has several, very interesting meanings and can be represented in several ways, e.g. thirty eight or XXXVIII or 111000. The data 38 can be an age of a person, a number in a lottery, a shoe number in Europe (7.5 shoe size in US), number of items, kilograms of potatoes, centimeters for the width of a table, therefore a data can been given an unit.

A code can have different business contexts, for example code AU can be:

- a two-letter ISO country code for Australia
- an element in the periodic table for Gold
- a unit of length for Astronomical unit.

The data 2001-09-11 with yyyy-mm-dd format or short 9/11 in USA is telling us the type of data as a date, and data 08:25 as a time. When we see a time in a clock, it contains an information but we can't be sure, if it is current time in our location. The data 07-08-09 is hard to convert to a date data type without knowing the format.

»Let's meet at half five« means 5:30 pm in Ireland and 4:30 pm in Germany or Denmark. Data needs a representation that is understandable to everyone. It is call conformed data with a process of conforming data to a common understanding.

A week number is not the same all over the world, therefore we have a ISO 8601 definition for week number 1 that is the week with the first Thursday of the Gregorian year (e.g. of January) in it. »Opening in Week 23 2015« is in Europe from 2015-06-01 to 2015-06-07 because week 1 starts at Monday December 29, 2014 with first Thursday at 2015-01-01. In USA Week 23 2015 is from June 7, 2015 to June 13, 2015 because week 1 starts at Monday January 4, 2015, source reference.

Contextually awareness is important, which needs to be accompanied with context, rules, intentions, etc. When data is brought into a context, it becomes information to which the representation is subordinate, as it is given the same meaning.

Information comes from Latin informare, which means »to give form«, that by information is understood the meaning content, a person attaches to data based on an adopted conversion.

When I know many of my friends' shoe sizes together with other information I have knowledge that I can use for a better birthday gifts or sell to a marketing campaign.

Only seeing the actual data in context will it mean something and provide information, or even tell you the story.

Lars Rönnbäck said that 38 alone is not data until it is distinguishable from noise, therefore 38 is noise due to misunderstanding. Putting 38 into context, like 38 kg. of potatoes, gives 38 an intent. Intent is what defines data. If some intent, somewhere, gave rise to 38 it's data. If not, it's noise. When seeing 38 in a column in a row of a table in a database, it's there with an intent, and then it's very unlikely to be noise. When data needs an intent, what does information need? Rönnbäck doesn't see a need to make a distinction. If it's a certain amount of context that suddenly makes something information that threshold is bound to be fuzzy and hard to properly define. If we see a tree whose branches happen to resemble the digits 3 and 8 next to each other, it's much more likely to be noise. Rönnbäck likes Shannon: A Mathematical Theory of Communication.

Metadata is in short »data about data« because metadata provides data or information about »other content data« (in danish indholdsdata) e.g. author and title of a book, table of content in a book, index in a book or an error description from a IT system like »violation of a foreign key constraint«.

Another example of metadata to the person weight 79 kilograms could be the date and time it was measured. When your camera in your smart phone takes a photo, metadata about date and time and location is stored in the image file.

Example of a XML structure with an element as SalePriceAmount with a data value content of a real sale price amount together with a metadata attribute as a currency or an unit:

<SalePriceAmount Currency="USD">295.73</SalePriceAmount>
<SalePriceAmount Currency="DKK">2236.14</SalePriceAmount>
<SalePriceAmount Currency="EUR">300.65</SalePriceAmount>

In a data warehouse metadata adds an extra value to the raw data from a source systems together with metadata for the data flow between data areas.

I am using **metadata columns** for housekeeping columns, support columns or administrative columns and for timeline metadata columns ValidFrom and ValidTo.
I have chosen to use two specific dates as metadata values (yyyy-mm-dd):

- Beginning of time as the first occurrence of data specified in column
  ValidFrom begins with date **1900-01-01**
  can include a time 00:00:00.0000000 (00:00:00.000 or 00:00:00).

- End of time as the last occurrence of data specified in column
  ValidTo ends with date **9999-12-31**
  can include a time 23:59:59.9999999 (23:59:59.997 or 23:59:59).

ValidFrom and ValidTo represents the span of time when a row in a table was the »current truth« in an active period as a time window for a value in a given state.

ValidFrom and ValidTo columns store the effective dates of validity for a row in a table, e.g. a Kimball type 2 dimension table where beginning of time in ValidFrom represents the initial row with 1900-01-01 and end of time in ValidTo represents the current row with 9999-12-31.

If a source system has a date column with a null and the null doesn't means »missing« and a data warehouse only like to have a non-null, then the data warehouse can use a forever perpetual date such as 9999-12-31 to identify a future undetermined date, an infinite date (in danish forevig uendelig dato for en uafklaret ikke aftalt dato). The benefit with a non-null date column is that a normal sql query join can be used to a calendar table or to a date table in a data warehouse. For an amount, a data warehouse can use a fixed value to represent a non-existent value.

Other examples of audit and metadata columns for housekeeping in a data warehouse is: Latest flag or IsCurrent, IsDeleted, IsInferred (artifact values).

I prefer audit columns and metadata columns with a suffix _audit and _meta.

Some data engineers like prefix such as DW or Meta_ for housekeeping columns.

A data warehouse use metadata e.g. to define and specify, describe, classify, trace, debug, audit, logs and metrics data, access restrict and monitor data.
Metadata is partly business data (e.g. business descriptions, trace/lineage, access restriction) and partly non-business data (e.g. monitor data, logs, metrics, technical definition).

Kimball has listed three key categories of metadata in a data warehouse solution:
- Business metadata describes the contents of a DWH including data lineage.
- Technical metadata describes the objects and processes in a DWH and the transformations (including business rules) for example for data integration, data conforming and data enrichment and calculation from section 1.1.
- Process metadata describes the results of the ETL process in a DWH, execution data or error data, start time, end time and rows processed. Read more in section 6.3.5. Similar metadata is generated when users query a DWH. Input to a performance monitoring and an improvement process.

Read more in Kimball Design Tip #75 and Kimball Design Tip #170.

Metadata columns will be mentioned in many places in the article and about its naming in section 6.3. I will end with some categories of metadata (auditing data):
Structural metadata is containers of data how compound objects are put together e.g. rule of a sql left-join and an enrichment of a derived column in a sql case-when statement.
Descriptive metadata is about a resource e.g. a link to a User Story Id in Jira and to a description and explanation in Confluence.
Reference metadata is about the contents, when and how it was created, record source, data lineage, movement of data between systems and data areas. When we deliver information to the business users, we must be able to tie that back to a source data set.
Administrative metadata is about how to read the contents e.g. column PerDate. Also permissions to the contents.
Process metadata is about how a ETL process is running e.g. a package start/stop time, its duration, execution result and error message.
Audit metadata is about reconciliation check e.g. rowcount, summarized measures to monitor and control movement of data between systems and data areas. Also about monitor and control of data quality and audit trail.

Statistical metadata is based on process metadata and audit metadata to describe the status of a data warehouse to be shown in a dashboard for Data management and DataOps.

Conclusion: It takes metadata to manage data and to use data.
More reading.

Auditing data is to track types of changes when data is inserted, updated and deleted, a user who made a change, an application or a ETL process that made a change.

I am using **audit columns** for InsertTime, UpdateTime, InsertExecutionLogId, UpdateExecutionLogId and for data lineage in column RecordSource.

Data auditing or data risk management is a comprehensive assessment of all aspects of data gathering, storage, and usage, including internal data such as financial records and external data like customer and market trend information.

Read more about data quality in section 1.14 and about an Audit dimension in section 4.4 to tag a wrong data row in a target table.

1.6. Transaction definition

Transaction
If Shakespeare had been a data modeler, I wonder if he had made the question: »A transaction has relationships or a transaction is a relationship?«

- If a transaction can be referenced to and it has a TransactionId as a primary key and it is a thing (its own entity) then it has relationships.
- If a transaction only contains foreign keys and it has no need for a unique identifier and nothing else will reference to it, then it is a relationship, e.g. many-to-many data-bearing relationship.

In Peter Chen's Entity Relationship Diagram ERD there are many kind of transactions e.g. an order, an invoice or deposit money into a bank account that is a many-to-many data-bearing relationship in the conceptual level design with the caption like »deposit transaction«. Moving on to a logical level design is when a many-to-many relationship becomes an entity and primary keys becomes foreign keys. Moving on to a physical level where an entity becomes a table in the database through a Relational Database Management System RDBMS. The most important thing for me is, that the names of tables and columns reflect as much as possible the user's conceptual world (in danish afspejler brugerens begrebsverden).

A counterpart to an invoice is called a credit memo, credit memorandum or a credit nota to reduce the amount of the invoice for several reasons, for example, the item is returned, the billed unit quantity is larger than the one delivered, the invoiced unit price is higher than agreed or there are errors in the delivery that justify price reductions. If a transaction in a source system is changed, corrected or cancelled then a data warehouse can calculate a counterpart transaction with a negative sale amount. The calculation of the final amount will be a summary of transactions whereby the final total amount will be less than the original amount.

Top 10 Analytics And Business Intelligence Trends.

1.7. Data areas in a dimensional modeling architecture

A description of the content of data areas or data layers or data zones of the throughput from multiple source systems to business users PC screen for improved analytic decision making. I am using the term **data area** to point out that an area may contain several databases to split data from multiple source systems to make the entire data warehouse solution with a focus on consistency, flexibility, scalability, efficiency, usability, reliability, auditability (consistent, flexible, scalable, effect, use, reliable, audit) and can be distributed over multiple servers. Each data area is labelled with a acronym formed from the initial letters.

Data flow

Source system→Landing zone area→Input data area→Archive area→Data staging area→Data mart area

SSA → LZA → IDA → ARA → DSA → DMA

Source file/table → Input table → Archive table → Staging table → Target table (destination)

1.7.1. Source system area – SSA (Operational system and Legacy system)

A company or an organization is using several operational systems to handle daily data of many different kinds. Most systems is based on a database with primary keys and foreign keys, and there is three types of candidate keys to identity an entity object, we can call them source keys that give us uniqueness in data as a unique identifier of a row in a table in a database.

Natural key exists in the real world e.g.:

- Fingerprint
- Country name
- Two-letter ISO country code and other codes that a user understand
- Three-letter currency code
- Brand name e.g. Louis Vuitton or Ford
- Famous football club e.g. FC Barcelona or Manchester United FC
- Phone number
- Hotel room number
- JobCode e.g. MNG is manager, PRG is programmer and CLN is cleaner
- Website address/url homepage
- An address with country + state/providence/region/district + postal code + street name + house number + floor number (stairwell, door number, PO box)
- Names of my friends combined with their Date of Birth (D.O.B.).

A composite natural key of multiple columns is still a natural key.

A natural key value is mutable (changeable, not immutable), and is meaningful for a human being. Back in 1990s we could remember many phone numbers, but the cell mobile phone contact list took over.

Business key exists in a system and is build as an artificial number e.g.:

- ISBN International Standard Book Number
- Vehicle identification number (VIN)
- Vehicle registration number plate
- Customer number
- Loyalty card number
- Purchase order number
- Invoice number with Product numbers
- Cadastral number or Cadastral reference or Parcel number for a property
- Insurance policy number
- Flight number
- General ledger account number
- Login name
- Social Security Number (SSN) a civil registration number like each danish person by birth is given a CPR number for Central Person Register (Person Civil Registration Identifier) but it is GDPR sensitive because it contains a natural part with a Date of Birth and the last digit tells gender as an even number for woman and odd numbers for man.

Some of the above numbers do exists in a natural way in the real world because they are printed on papers and things, therefore can be called an application key.

A composite business key of multiple columns is still a business key.

A business key value is either mutable (changeable) or immutable (unchangeable), and is meaningful for a human being where a business user prefers to use a business key value to identify an entity for a search criteria lookup value giving in a phone call to a company, a bank, a hospital or the government.
A good business key should remain constant over time and not change frequently, stability.

Surrogate key exists in a system with the sole purpose of uniquely identifying a row as an artificial auto-generated unique sequence number or an identity column (id, uid unique identification) that is used as an unchangeable column instead of a natural key or a business key or a composite primary key.

A surrogate key column must not allow null, must have a constraint **not null**.

A surrogate key is represented as an integer or as a code or as a GUID.

A surrogate key value is immutable (unchangeable), and is meaningless (aka not-meaningful) for a human being and will normally not be exposed to outside business users, but is useful for join operations inside an IT system. A surrogate key value is not influenced by any business rules, therefore it is often an integer value, because a hashbyte value is dependent of the data and can cause hash collision and no guarantee to be unique.

Surrogate key as an integer it not idempotent compared to a hashbyte value, because there is no guarantee of the same integer value to an entity instance after a rerun.

A natural key or a business key is what the business uses to uniquely describe data. If a natural key or a business key is immutable and unchangeable in a source system then call it a durable natural key or a durable business key.

A natural key value for the state Wisconsin is WI, and a business key value is 30 source or is 49 because it is assigned in alphabetical order, and a surrogate key

value is 17 a random number. What if one day Wisconsin decides to change its name to New Australia? The WI, 30 and 49 will be obsolete and the 17 still stands.

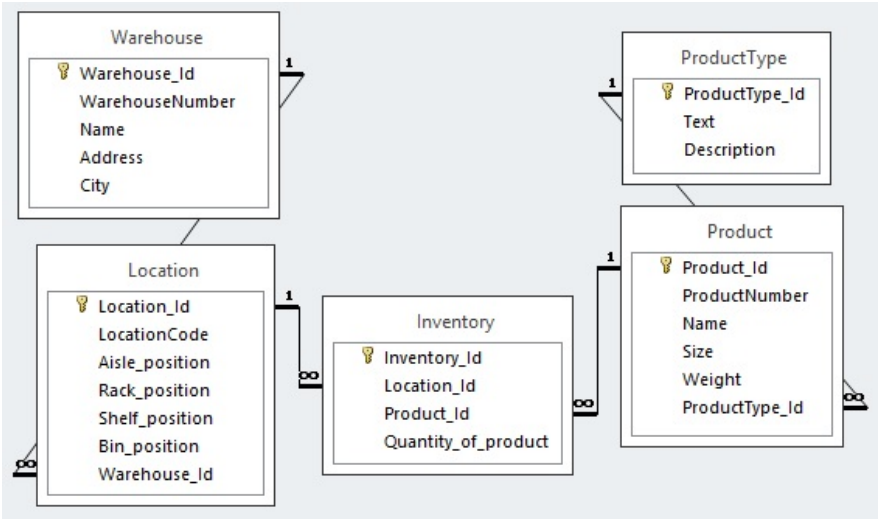
A table called Salestype has a business key as an one-letter code and has a natural key as a textual value, e.g. R = Retail sales, D = Direct sales, O = Online sales.

A table called Status has a surrogate key StatusId as an artificial auto-generated unique sequence number or an identity column, a column StatusNumber as a durable business key and a column StatusCode as a durable natural key with a textual value which can be understood by a human being, otherwise look at column StatusName which does not need to be unique.

A table called Customer has a surrogate key CustomerId, a column CustomerNumber as a business key and a column CustomerName as a natural key.
It is common to have several customers with same name, therefore I will not call CustomerName a durable natural key. It is common in a customer system that the same person exists several times because a person contains in duplicate rows with the same name and address but with different CustomerNumber, therefore I will not call CustomerNumber a durable business key.

From a data warehouse point of view, a natural key and a surrogate key is also a business key. For example, a table called LeadState has a surrogate key Id and a column State with values as »New, Contacted, Qualified, Lost« then a data warehouse lets Id become the business key because a text value can be changed over time e.g. »Lost« will one day be changed to »Not interested«. A beloved child has many names.

Example of a ERD with surrogate keys and business keys
Each entity has its own surrogate key as the primary key used in relationship as the foreign key, and the surrogate key values are source-specific and meaningful only within one source system.
Each main entity has its own business key as a unique identifier which is WarehouseNumber, LocationCode and ProductNumber and we analyze and find out that the business keys values are cross-source and meaningful across source systems.



Since a data warehouse solution like to integrate and merge different source systems, we cannot use surrogate keys for that, because they will have different values in the other systems. This means that either a source system must join the tables to provide the business key WarehouseNumber in Location table and the business keys LocationCode and ProductNumber in Inventory table, or else a data warehouse data staging area will do the joins.

For entity ProductType there is no immutable (unchangeable) business key because a value in Text column can change over time. Therefore we will use the surrogate key column ProductType_Id as the business key in a data warehouse solution.

Key mapping table from Kimball ETL Toolkit page 48 and The Kimball Group Reader from 2016 page 622 is also called surrogate key mapping table, source key mapping, cross reference table, support table, translate table or shadow table, is a table for each business key or composite business key of multiple columns, (or natural key) to transform business key values either to an immutable surrogate business key, e.g. a code which can be understood by a human, or to an artificial auto-generated unique sequence number or an identity column as a **durable key** for later use in a dimension to glue multiple rows together in a timeline for a type 2 dimension or a type 7 dimension.

A durable key is a surrogate key of a business key, because a business key value can be a mix of number and text or can be a composite business key with values in multiple columns.
A durable key column must not allow null, must have a constraint **not null**.
A durable key column is represented as an integer or as a code.
I prefer a suffix **_dkey** for a durable key column in a key mapping table and in a dimension table.
I prefer a suffix **_bkey** for a business key column(s) in a key mapping table and in a dimension table.

A durable key value is immutable (unchangeable), therefore a business key value or a natural key value has the same durable key value throughout the life of a data warehouse solution. Never empty or truncate a key mapping table.

Alternative names are durable supernatural key, durable surrogate key, stable key, persisted key, immutable key or stable identifier.
(Durable in danish holdbar, varig eller slidstærk).

Read more at Kimball page 46-47 and article at kimballgroup.com.

A key mapping table has often only one dimension table attached, because a dimension does fetch and integrate data as a result of a merge of multiple source system data and become a conformed dimension and a shared dimension.

Since a dimension can have multiple source data, a key mapping table contains a business key for each of its source systems for data lineage to provide only one immutable surrogate business key e.g. a code or a durable key across multiple source systems.

A business key with sensitive data for example a Social Security Number (SSN) can get anonymization or pseudonymization through a key mapping table, and a data warehouse solution use the durable key SSN_dkey.

Example of a key mapping table with a business key and a durable key
A product registration system has a business key called ProductNumber which is unique for each product. A key mapping table has a business key column (suffix _bkey) with the values from ProductNumber where each business key value is translated to an integer value in an artificial auto-generated unique sequence number or an identity column in a durable key column (suffix _dkey).

ProductNumber_bkey	Product_dkey	Business_context
245600390	1	PRS | ProductNumber
571145292	2	PRS | ProductNumber
571145311	3	PRS | ProductNumber
746153022	4	PRS | ProductNumber

A constraint non-clustered primary key or a constraint unique key on column ProductNumber_bkey to ensure uniqueness and no duplicates and to speed up to fetch a durable key value from a business key value.

Example of a key mapping table with two business keys and a durable key
European customs authority uses an EORI Economic Operators Registration and Identification number as a unique identifier for each company for import and export of goods. In most source systems a EORI-number starts with a two-letter country code for the company's home country, e.g. DE for Germany, DK for Denmark and FR for France, but few source systems have omitted a country code, therefore a key mapping table has two columns for two business key values to assign one durable key value for same-as.
Business key column has suffix _bkey. Durable key column has suffix _dkey.

EORI_bkey_country	EORI_bkey_without_country	Company_dkey
DE8472463	8472463	6459
DK49562	49562	8473
DK76034	76034	8477
FR123456	123456	9948
FR123456	123457	9948

The last row shows a mistyping or a typos in a source system, and a data steward has matched EORI 123457 together with FR123456 in the key mapping table to the same durable key value 9948, a true integration of business keys and their values.

Example of key mapping with multiple business key values to one durable key value
A key mapping model based on two tables:
1) A main table contains all business key values from different source systems in a business key column (suffix _bkey) with a data lineage column, and each business key value is translated to an integer value in an artificial auto-generated unique sequence number or an identity column in a durable key column (suffix _dkey).

Customer_dkey	Customer_bkey	Business_context
1	176	CRM | CustomerId
2	359	CRM | CustomerId
3	421	CRM | CustomerId
4	HA-65	ERP | CustomerNo
5	JW-11	ERP | CustomerNo
6	MB-01	ERP | CustomerNo
7	SJ-08	ERP | CustomerNo
8	ML-08	ERP | CustomerNo
9	574	CRM | CustomerId
10	MS-34	ERP | CustomerNo
11	JD-66	ERP | CustomerNo

2) A relationship table contains all relationships among the business key values that is represented with durable key values from the main table.
For example CustomerId 176 and CustomerNo HA-65 is the same customer in two source systems CRM and ERP, and therefore get the same-as durable key value 1 in the relationship table below.

For example CustomerNo MB-01, ML-08 and MS-34 is the same customer in the source system ERP because it changes the business key value when a customer changes their name, e.g. Marie Beaulieur, Marie Lesauvage and Marie Sainte because she has remarried and took her new husband's surname. Therefore get the same-as durable key value 3 in the relationship table below.

Sometimes a key mapping table needs to support time variability with the timeline metadata columns ValidFrom and ValidTo to have a time period or time span for a mapping to show an active period, because a source system might reuse a business key value, as happened for CustomerId value 574 with durable key value 9 that is used for two ERP customers CustomerNo SJ-08 and JD-66 in different time periods.

The relationship table column Customer_dkey_2 contains one row for each durable key value from the main table where column Customer_dkey_1 including itself and another source system when it has a corresponding business key.

Customer_dkey_1	Customer_dkey_2	ValidFrom	ValidTo
1	1	1900-01-01	9999-12-31
1	4	1900-01-01	9999-12-31
2	2	1900-01-01	9999-12-31
2	5	1900-01-01	9999-12-31
3	3	1900-01-01	9999-12-31
3	6	1900-01-01	2007-11-15
3	8	2007-11-16	2014-07-14
3	10	2014-07-15	9999-12-31
7	7	1900-01-01	9999-12-31
7	9	1900-01-01	2020-12-31
11	9	2021-01-01	9999-12-31
11	11	1900-01-01	9999-12-31

For a dimension table, I use the first durable key, Customer_dkey_1, to be present.

When a data warehouse receives a business key value, e.g. CustomerNo JW-11 from a source system ERP, a ETL process will first perform a lookup in the main table in column Customer_bkey to fetch a durable key value in column Customer_dkey, e.g. 5. Next perform a lookup in the relationship table in column Customer_dkey_2 to fetch a durable key value in column Customer_dkey_1, e.g. 2.

[Inspired by the party data model as a universal data model with a Party entity and a PartyRelationship entity with a type of relationship with values, e.g.: same-as, derived from, member of, defines, matched by rule X or with a confidence factor or fuzzy identity.]

A key mapping table must be able to handle a business key value which can be mutable (changeable) to keep the same-as code or same-as durable key value. For example, a business key CustomerNo does change when a customer changes name, or a business key Employee number may be changed if an employee resigns and some years later is rehired, the belonging durable key value must be remained.

Kimball calls it a durable supernatural key and it can be part of an integration hub in a data warehouse system. Durable keys are mandatory for dealing with ambiguities in the source system business keys (natural keys). Creating and assigning durable keys allows you to work around business changes to the business keys, or to integrate duplicate or disparate data. But durable keys are just the start; there is a lot more to data deduplication (deduplicating) and data integration. (Kimball Design Tip #147).

Kimball page 101: »The durable supernatural key is handled as a dimension attribute; it's not a replacement for the dimension table's surrogate primary key« which

is called a dimension key regardless of dimension type 0 – 7, see later at Data mart area.

Multiple source systems have a business key for a Social security number per person with a primary key as a surrogate sequence number, therefore we can't use the primary keys to interconnector (join, drill-across) the systems data, instead we use the business keys. The value of a business key can change over time e.g. a person obtains witness protection and gets a new Social security number, therefore it is up to a ETL process to make a **solid mapping**. When a business key is an employee number and an employee resigns and some years later is rehired, there could be two different employee numbers for the same employee (EmployeeNumber or EmployeeId), and a data warehouse must find a way to map (»glue«) them together, so the employee only occurs once in a dimension with help from a key mapping table that remain the durable key. The other way around can also happen in a source system where the same value of an EmployeeNumber or an EmployeeId is used for two different employees.

A key mapping table will be used at various points in a ETL process:

- Transform a business key value to a durable key value while loading to a staging dimension table, later will be merged into a dimension table.
- Transform a business key value to a durable key value while loading to a staging fact table, later will be merged or partition switch into a fact table.
- Handy while snowflaking source data into a snowflake dimension.

I like to implement a key mapping table as part of a Usage supporting database in its own database schema Keymap or by including Keymap in a key mapping table name. When there is no Usage database, a key mapping table will be part of a Data staging area.

More key mapping table examples in a later paragraph »Data integration mapping by acronyms example«.

An alternative to have multiple business key columns for each source system in a key mapping table, Kimball introduces a single cross reference table with columns, e.g.: Data Warehouse Natural Customer Key is a special natural key created by the data warehouse, Source System Name and Source System Customer ID in Kimball Design Tip #67.

Data profiling of a source system is very important and the choice of column to become a data warehouse business key requires a good data analysis and criteria for the choice and data type and data range.
Data profiling of source data to a data warehouse is a data preparation task to ensure a data understanding, describe its content, consistency, structure, data type, format, null, business key, etc. and maybe doing conversion, cleansing, correction, e.g. removing duplicates and remove data that isn't accurate and/or isn't relevant. Write it all down in a data profiling document to help the development of ETL pipelines. Read more. Some data profiling quotes and figures.(Kimball Design Tip #59).

Be aware of a value of a natural key or a business key can be changed in a system because the original value was entered incorrectly and need to be corrected. A composite natural key in a Member table with Name+Address will be updated when a person change place of residence. I saw in an airport, that the flight number was selected as the business key for the data warehouse and used in a ETL process, but no one had taken into account, that when a plane was delayed, the letter D was added at the end of the flight number, therefore the data warehouse thought it was a new flight.

A surrogate key as an integer or a guid won't ever need to be changed, but if a customer is deleted by a mistake and the customer later will be inserted again with the same CustomerNumber as business key, then it will get a new sequence number in CustomerId. When a company has several operational systems handling customer data, each system will use its own surrogate key for CustomerId, therefore business key CustomerNumber is the glue between the systems.

It is important for a data warehouse to receive a value for a business key (and/or the natural key) from a source system, and in case of a value has been changed, that data warehouse also receive the before-value. A database has many tables for example, a Customer table has columns of CustomerId as surrogate key, CustomerNumber as business key and a CustomerName, and an Invoice table has columns of InvoiceId, InvoiceNumber, InvoiceDate and CustomerId as foreign key, the invoice data must be extended with the business key CustomerNumber either by a source system or inside a data warehouse to know the customer of an invoice.

I prefer that a data warehouse also store the value of surrogate key to be helpful when a value of a business key is changed and to obtain traceability for data lineage to tell where does the data come from, more in section 1.15. Merging data and integrating data can be hard to set up in a ETL process and to test afterwards.

For a dimension in a data mart, it is important to make a data profiling of a source system or an operating system to achieve an understanding of how data values are changing, and to know whether or not a data value will change or will never change over time in a source system, because it may choose a type for a dimension.

When a data value is changing it is good to have together with a business key a business date from a source system, e.g. ordered date, purchase date, shipped date, delivered date, receipt date, invoice date, transaction date, event date, wedding date, to indicate a date of change of one or more data values connected to the business key to get a time-variant data warehouse. A date of change audit column is very useful for a Kimball type 2 dimension to determine a new value in timeline metadata column ValidFrom, read more in section 6.5.2.

1.7.2. Landing zone area – LZA

From source systems to landing zone area as a file storage of source data where a landing zone area can be a file server. A landing zone area is a data reception area with data from mutiple source systems. A data set can arrive in many formats, therefore a good **data profiling document** is nice to have. For example:

- csv file (comma separated values), txt file, xml file, json file, parquet file.
- Excel file or pdf file.
- gz file or zip file with compressed file(s).
- Database file e.g. Access accdb file or mdb file.
- Database backup file from a SQL Server, DB2 server, Oracle server, etc. and a backup file can be restored at a landing zone area server.
- Database data is copied from a source database server to a landing zone area server persistent database placed in a dmz or a similar zone:
- by database replication (batched)
- by data streaming (streamed) where data flows out from a source system through a delivery or a redelivery
- by receiving data from a real-time data replication that fetch new, changed and deleted source data. An example of a Change Data Capture CDC tool is Qlik Replicate Change Data Capture, read more in section 1.9.

A landing zone area can be a file server to receive files and to store files.
Sometimes called a Blob Storage.
A file can be push from a source system to a FTP server or a file server can pull a file from a FTP server as mentioned in section 1.9 Data capture or data ingestion.

A landing zone area can be a database server to receive a database backup file that will be restored to a database at the server, or there is a persistent database that receive data from a source system.

This area is also called Operational Data Store (ODS), a place to store source data or Transient zone or Ingestion layer. Microsoft is using the term Landing Zone as the place where source data is landing and often is never deleted or deleted after a longer period.

A file it can be transferred to a file handling area, or a file can be fetched from a SFTP server through WinSCP to a folder which represents a landing zone area, and from there copy the file to an archive folder to keep the original source files, and move the file to an input folder for later unzip to another folder and loading the file into a data warehouse. A file can have metadata in the filename like a date of contents and delivery. Sometimes a delivery consists of several files which can be advantageously placed in a delivery folder with a name including a date value or a batch number. Contents of the files must be checked for data quality before they are loaded into the input data area. In case you like to open a 20 GB csv file and delete the last line that is corrupt, use Sublime Text, its free and no registration, only long waiting for large file. A nightmare is like 25 csv files that occasionally change filenames and change column names and data types and content.

It is an advantage to receive a delivery note file (metadata) together with data files from a source system with information about the data files like filenames, maybe a missing file delivery and the number of lines in a data file or rows in a xml file. For a database backup file it is a must to identify them as full backup or incremental backup before restoring and the datetime for the backup.

A data set should have accompanying metadata describing the origins and processing steps that produced the data for tracking data lineage.

Receiving data in a xml file, a data profiling is in a xsd file XML Schema Definition.
Receiving data in a csv or txt file, a data profiling is in a Schema.ini Read more.
A data profile for a csv or txt file must determine different formats, for example:

- Date format, e.g. 05/08/2014 what is order of day and month as format mm/dd/yyyy or as format dd/mm/yyyy.
- Amount format with a decimal separator and a thousand separator as a period or a comma, e.g. 1,234.56 or 1.234,56, how about scientific notation.

- Boolean (bit) format True and False, T and F, 1 and 0, Yes and No, Y and N. Sometimes called a Flag indicator. Sometimes with a null.
- Code page, Encoding, Byte Order Mark (BOM).
- Header row with column names in the first data row, Header row delimiter, Row delimiter, Column delimiter, Text qualifier, Field terminator.
- Illegal characters where a  diamond with a question mark in the middle (rhombus) is a replacement character, a character place keeper, to display whenever a character is not recognized in a character set encoding instead of display an unknown, unrecognized, or unrepresentable character as a blank space.

I prefer a csv or txt file with encoding UTF-8 (codepage 65001) and a column delimiter as Tab as a list separator <Tabular>, Chr(9), \t as a Tab delimited file.

1.7.3. Input data area - IDA

From Landing zone area to Input data area for temporary storage of source data which has been received in files as csv, xml, json, parquet, excel, access, database files, database backup files or by database replication, data streaming or CDC.

The files in landing zone area can be imported or extracted into a input data area to become database tables with loose data types e.g. string, text, varchar or nvarchar with a length of 100+ characters to process raw data as gently as possible.

When a source system has millions of transaction data it can take a long time to do a full load every night, therefore to save on loading time from a source system to a data warehouse it is good to detect delta data as new data, changed data or removed data in a source system. To fetch data from a source system database by using incremental load or delta load, and mostly it happens from a Input data area without a Landing zone area. Read more in section 1.9 Data capture or data ingestion.

This area is also called Raw data area, Data Sourcing layer, Source, Legacy, Extract, Capture Operational Data layer, Data Acquisition.

Empty input data area in the beginning of a ETL process.

Data types will be adjusted to fit with the receiving database system, and it is a good idea to use a wide data type e.g. a source system provides a string of length of 6 or 50 characters, I will make it as 100 characters to be prepared for a later extension of characters in a source system.

Do reconciling between source systems and input data area with reconciliation of row count and sum of values and mark as reconciled and do auditing report (to reconcile in danish at afstemme, stemmer overens). Maybe a receipt system where IDA tells a source system that data has been received. Data might be wrong in input data area or in archive, and later on a source system will resend a new revision of data. Read more about data reconciliation in section 1.13.

For the data lineage most tables in the Input data area will include audit columns: RecordSource as a reference back to a source system e.g. value "Dynamics-365.Sales.Product". RecordSourceId is origin primary key column from source system. IdaBatchDataCaptureId is in all rows in all tables and will use the same number of identification together with a IdaBatchDataCaptureTime as a received date and time to represent a snapshot of the total amount of data as it look like in a source system, and can be used later for keep history and in ETL jobs/packages/transformations/sql will reuse the same id and time for all data rows. IdaInsertTime (datetime2(7) default sysdatetime(), not unique per row) when the fetching is done. Sometimes a source system has a Changed column with data/time of first inserted or data/time of latest updated data in the row/record that can be used for later history. Read more about data lineage in section 1.15.

Exchanging data from a source system to IDA can be in different formats e.g. JSON data-interchange format where I prefer storing data for IDA in a relational database with tables with rows, columns and data types, I must parse a json format.

Kimball recommended that a source system express data at the lowest detail possible for maximum flexibility and extensibility to provide the data warehouse for simplicity and accessibility and it will be the data warehouse to make summary data, and not the source system. IDA is a mirror of source data.

I recommend for each source system to create its own Input data area e.g. IDA_Dynamics365, IDA_Salesforce, IDA_ServiceNow, IDA_Proteus to keep the source data separately. Each IDA has metadata for scheduled expected delivery with dates, file names or table names, estimated number of rows, receiving time, row count and flags like IsReceived, IsCompleted, IsReady together with different control tables, reconciliation tables and staging tables to make data ready for the archive area.

I recommend for each source system to create its own ETL process (job with packages) to extract, maybe a little transform and load data into an Input data area and possibly furthermore to an Archive area. When a source system is bad or down, data must continue to be loaded from other source systems. When a daily source data delivery is bad, the next day delivery must still be able to be loaded and it is nice when a redelivery of a bad data can be provided and handled in a ETL process.

1.7.4. Archive area - ARA

From Input data area to Archive area for keeping source data indefinitely through forever storage of source data from source systems as a data basis for a data warehouse.

This area is also called Persistent Historized Data Store, Persistence Layer, Storage layer, Detail Data Store, Data Repository, History or Archive data area.
Archive is a versioning of the source.

Never empty archive area because it is archiving of time-variant source data and it will retain and preserve historical value changes in the source system.

Simple data adjustment can be done to gain same date and amount format and same data representation of a social security number, but it will be in new columns so the original values are unchanged.

For the data lineage most tables in the Archive area will include audit columns: RecordSource, RecordSourceId, IdaBatchDataCaptureId, IdaInsertTime and ArcBatchDataCaptureId, ArcBatchDataCaptureTime, ArcStorage, ArcInsertTime (datetime2(7) default sysdatetime(), not unique per row), ArcTs (timestamp, rowversion, unique per row), ArcRecordId is a unique sequence number per row per table to obtain data lineage and traceability back to the archive and the column can also be used to fetch and pull delta data out of the archive that have not yet been loaded into the data warehouse. ArcGlobalId is a unique sequence number per row across all tables in archive.

A different kind of an archive that does not do »forever storage of source data« can be called Current Data Area – CDA with the purpose to have a storage of current source data because the data change frequency and volume of changes is too big to retain and preserve historical value changes in a source system, therefore the archive has »current only« version of source data to improve performance and use less storage. CDA means that you have to detect data to insert new data, to update changed data and to delete removed data from a source system. CDA can be based on either full load, incremental load or delta load. CDA does not mean that you can't have a type 2 dimension with history in a data mart area.

Read about archive in chapter 8.

1.7.5. Data staging area - DSA

From Archive area to Data staging area for extracting archive data by full load, incremental load or delta load for transforming data enrichment according to business rules to identify dimension data, dimension values and fact data to staging tables to shape and suit tailored target tables in a data mart, for ensuring that data is trustworthy in order to prepare data for loading into a data mart table for the purpose of querying and analysing in different ways by the business users for their analysis, reporting and dashboards.

Data processing can be for example:

- cleansing, cleaning, scrubbing read more about data cleansing
- fetch inaccurate or incompatible data, inconsistent formats and outliers
- filtering, checked for redundancy, removing duplicates, purging
- filter wrong data and error data away for making correct data to next area
- handling of wrong data or error data into a wrong table
- ensuring of referential integrity and business key or application key
- handle missing values, unknown values, add default values, correct errors
- validating, validation, auditing, authenticating for syntax and business rules
- conducting audits to ensure data quality and compliance with business rules
- translations, mappings, mapning, conversions, convertings, formatting
- combining, integration, integrating, integrate, integrated, merging, merge
- establish consistency, consolidating, transposing, ordering
- conforming data and renaming columns to common names and terms
- determine granularity and grain
- enriching data by adding derived value in new column in pipeline expression
- calculation, computing, aggregation, summarization, calculated measure KPI
- common units of measurement for measures and metrics, standardizing
- encrypting protecting data governed by industry or governmental regulators
- handling of key mapping table for business key integration and durable key
- handling of data lineage to an audit column RecordSource
- handling of business date to an audit column DateOfChange

- handling of corrected flag in source data to a metadata column IsCorrection
- handling of delete flag in source data to a metadata column IsDeleted
- handling of artifact values/inferred member to a metadata column IsInferred
- making data ready for a type of a slowly changing dimensions table
- making data ready for a group dimension table and a bridge table
- making data ready for a type of a fact table

Read about fact table and types of fact tables in chapter 3.
Read about dimension table, dimension tables and types of slowly changing dimensions and artifact values or inferred member in chapter 4.
Read about business date and date of change in section 6.5.2.

DSA can be divided into tables for StagingInput and for StagingOutput by using database schemas like StagingDim, StagingFact, StagingAnalysis (one big table) to shape and suit target tables in schemas Dim, Fact, Analysis in a data mart area.

This area is also called Integration layer, Integration area, Transformation area, Conformation area, Calculation, Enrichment, Prepare, Temporary, Work, Refined layer, Refined zone, Standardised, Staging, Stage or Stg where the core of the data warehousing process takes place to a unified data model.

Empty data staging area in the beginning of a ETL process because data is only transient and temporary in this area, except key mapping tables and wrong tables when they are implemented here instead of being a part of a Usage supporting database.

Data from a csv or txt file in Archive area where each column has a data type of string e.g. varchar(100), I will for each column do a try_cast **data conversion** to an appropriate data type, e.g. Date, Decimal(19, 4), Currency, Money, Integer, etc. according to a data profiling document to validate each data. If data in all columns in a row can be concerted, then I will insert the row into a staging table. If data in a column in a row can not be converted, then I will insert the row into a wrong table and send an email to the operation and let the ETL process continue.

Calculation and computing metrics, e.g. total cost and revenue with same unit, elapsed time and overdue time. A calculated measure for a target table in a data mart is common to have in a staging table, e.g. SaleAmount = UnitPrice x Quantity.

A Persistent Staging Area (PSA) is never emptied, because it is a collection of data from incremental load or delta load from multiple source data with integration (full outer join of some archive tables), preparation and cleansing, etc. before the data is loading (incrementally) into a data mart. Read more. See example in section 8.6. A DSA table can extract data from a PSA table.

Redundant data and **Removing duplicates**
Data deduplication refers to a technique for eliminating redundant data in a data set by selecting one of the duplicate rows and removing the other duplicate rows and insert them into a wrong table and send an email to the operation and let the ETL process continue. In the process of data deduplication, extra copies of the same data is deleted, leaving only one copy to be stored. The proces is called removing duplicates, because the unwanted data is discarded.

Examples of concrete **data cleansing** are trim string, unicode string, max length of string, null gets a value as 0 or empty string '', replace illegal characters, replace value, verification of data type and do data conversion to a suitable data type, e.g. a csv file contains a value 1.0E7 in the scientific notation and it will be converted to the value 10000000 and saved in a column in a table. Data syntax, date format like mm/dd/yy or dd.mm.yyyy will be converted to yyyy-mm-dd, correcting mistyping, typos or misspelling, fix impossible values, punctuation and spelling differences to achieve common format, notation, representation, validate data e.g. you expect two values »True« and »False« in a boolean (bit) column but suddently there is a value like »none«, »void« or a sign ? therefore you must correct data to »false« or send an email to the operation and let the ETL process stop.

Examples of concrete **data integration** from multiple source systems is to make data conformed, e.g. a gender code from one source system use »Man, Women« and another source system use »M, F« will be conformed to »Male, Female« through a mapping of source data.
Example of VAT Value Added Tax where one source system has a SalesRevenue as include-VAT and another source system has SupportRevenue as exclude-VAT, and an integration will calculate SalesRevenue as exclude-VAT to do a calculated measure for a TotalRevenue in pre-VAT.

When an archive is stored on an another server it is common to load data from the archive into a data staging area at a data warehouse server, e.g. data to a dimension, and from a dimension staging table perform a loading as a merge or an insert and update to a dimension table in a data mart area. Same is done for fact data.

I know that many like to send a query to a source system for translate names of columns and data types, merge or divide columns like a Name to become two columns of FirstName and LastName (or Surname), but then a data warehouse will not receive the original source data.

I like to use computed columns in a staging table for calculation, string manipulation. Sometimes I use a hashbyte value for a comparison column for a composite business key that is composed of multiple columns (ComparisonBkey_meta) and for the other data value columns (ComparisonData_meta) to merge and load data from a staging dimension table into a mart dimension table. Other times I use a generic stored procedure with dynamic sql statements (string gymnastics <template> merge construction) that will perform the load from data staging area to data mart area for many dimensions.

DSA database will contain different auxiliary tables with names as: stg, stage, tmp, temp, temporary, and with multiple staging tables to a target table, where the staging tables can have an additional number e.g.:
Stg_Customer_Order_1, Stg_Customer_Order_2, Stg_Customer_Order_3
and the last staging table to shape and suit the target table is using number 0 as in Stg_Customer_Order_0.

DSA will perform a data quality and filter wrong data and invalid data into a quality assurance and quality control (QA/QC) database. DSA continues reconciling data from the archive for referential integrity e.g. foreign key value exists as primary key value, relationship cardinality rules for dependencies as mandatory, optional or contingent to make sure there is a foreign key or not, it can be null or it needs a value. Many other value checks and summation and so on for a validation check of data. When all tables in this area is loaded with correct data, the DSA is successfully completed. A source system's new date or changed date or the IdaInsertTime or ArcInsertTime which represent a batch of insertion and mark all data to that batch time no matter which source data is coming from. In case of ValidFrom and ValidTo, all ValidFrom will be using the same batch time which is useful for join between tables and between source systems to fetch the right data shapshot at a particular time back in history. Read more about data quality in section 1.14.

Furthermore, derived columns are ready for loading into the next area of the data warehouse. In case of an error in a ETL process or in data there will be raised a halt condition to stop the ETL process. Other times data will pass by with a flag in an Audit dimension. Kimball has called it data wrangling to lasso the data and get it under control (data munging).

Remember to do an auditing reporting after a validation check.

For the data lineage most tables in the Data staging area will include audit columns from the Archive area as RecordSource, ArcStorage and ArcRecordId.

(In danish data staging area stands for et data forberedelsesområde eller data rangeringsområde til data vask, data udsøgning, data berigelse, data beregning, data behandling, data bearbejdning, data sammenlægning, sammenstilling, samkøring, tilrettelæggelse og klargøring af data hvor data er blevet vasket og strøget).

Kimball calls this area a ETL-system or the first system that is off limits to all final data warehouse clients and he use an analogous: »The staging area is exactly like the kitchen in a restaurant. The kitchen is a busy, even dangerous, place filled with sharp knives and hot liquids. The cooks are busy, focused on the task of preparing the food. It just isn't appropriate to allow diners into a professional kitchen or allow the cooks to be distracted with the very separate issues of the fine dining experience.« [Back room ETL system and Front room presentation area, read more.]

Kimball's ETL is using Data Staging Area (DSA) as a temporary space to create a data warehouse as a collection of data marts.

1.7.6. Data mart area - DMA

From Data staging area to Data mart area based on dimensional modeling with facts of measures, metrics, analysis variables surrounded by conformed and shared dimensions with context descriptive data, text, hierarchy, group, band to explain the measurements. A data mart is designed for a specific business case with specific subject data from one or more business processes.

Dimensional modeling ends up with a dimensional model such as illustration.

- Star schema with one fact surrounded by conformed and shared dimensions.
- Constellation schema with few facts sharing conformed dimensions.
- Snowflake schema with one or few dimensions have become snowflaking.

A data mart area will contain several data marts (short mart) each with their own specific purpose and tailored dimensions and facts that are business process orien-

ted and and fulfills business needs to answer key business questions and helping a stakeholder or a business user to make better data-driven decisions.

Data processing can be for example:

- handling for a type of a slowly changing dimensions table SCD
- handling for a group dimension table and a bridge table
- handling for a type of a fact table
- merge a staging table into a dimension table, group dimension, bridge table
- merge a staging table into a fact table, or partition switch into a fact table

A data mart with dimensions and facts make it easy for a business user to:

- do ad hoc query in SQL or through a BI tool for self-service BI
- search for current data as a contemporary snapshot
- search for registered data as happening, occurred, registered or took place
- give a historical date to fetch dimension and fact data at any point of time
- develop a report and a dashboard
- develop a Multidimensional model for a OLAP cube
- develop a Tabular model for a OLAP cube (OnLine Analytical Processing)
- delivery of data to a report or a dashboard through stored procedure
- delivery of data to a OLAP cube by processing
- delivery of data to other systems
- delivery of data to Data mining (DM)
- delivery of data to Machine Learning (ML)
- delivery of data to Artificial intelligence (AI)
- etc.

A Sales mart example with data values in section 1.10.4 and many other examples in this article about Dimensional modeling to be implemented in a data mart.

Keep historical data in a dimension and in a fact with timeline metadata columns ValidFrom and ValidTo.

A data mart can have multiple fact tables with different granularities.

One or multiple fact tables can create an extra **derived fact** table with special calculations and search filter criteria to enrich data to match the business requirements specification or the user story.

Make a conform name for a column with a conform data type, for example when you merge customer addresses from multiple source systems make a string extra long to fit all address characters and avoid a value to be mandatory or set a default as the empty string to ensure a robust ETL process.

Never empty data mart area and always backup before a ETL process.

Data mart is a front room for publishing the organization's data assets to effectively supports improved business decision making. A data mart contains one data area for one purpose and is subject-oriented, therefore a data warehouse will consist many data marts and they will have some common dimensions. To avoid to maintain the same dimension in multiple data marts may it be considered to have a dimension data mart that share its tables through views inside the other data marts.

For the data lineage most tables in the Data mart area will include audit columns from the Archive area as RecordSource, ArcStorage and ArcRecordId.

(In danish data mart area stands for behandlet lag, tilrettelagt lag, klargjort lag med data som opfylder forretningens data behov).

A data mart based on dimensional model with dimensions and facts can be called a Multidimensional database (MDB).

Kimball calls this area for a dimensional presentation area, and he use the term first-level data mart with base-level fact that contains detail level data from one business process and from one source system, and the term second-level data mart with aggregate-level fact or consolidated fact that contains overall level data from multiple business processes and from multiple source systems for a consolidated mart.

This area is also called Consolidation area, Information mart zone or Delivery layer.

A data mart does not necessarily have to use a dimensional model, instead it can be a relational model e.g. for a file delivery or to machine learning, or a graphical model with graph structures to model complex systems and relationships between business entities. Read more.

The overall purpose of a data mart is to fulfill business requirements specification or user story from stakeholders and business users with shaped and suited tailored tables in a data repository purpose-built based on a data modeling methodology for a data model to serve business needs.

Dimension key is a primary key in a dimension table.
Dimension key is a foreign key in a fact table.

Read about fact table and types of fact tables in chapter 3.
Read about dimension table, dimension tables and types of slowly changing dimensions and artifact values or inferred member in chapter 4.
Read about business date and date of change in section 6.5.2.
Read about dimensional modeling methodology and the four step process to design a business process-centric fact and report-centric fact in chapter 5.

1.7.7. Presentation interface area - PIA

From Data mart area to Presentation interface area through data access tool like Reporting Services (SSRS) and Microsoft Access, or data visualization tool **BI tool** like Excel, Power BI, Tableau, QlikView, Qlik Sense, Alteryx or Targit can import data from a dimensional model and handle in-memory calculated KPI as quick measure using filter.

The purpose is to help and support a business user for analysis and reporting to solve a business case.

A **OLAP cube** for **Multidimensional** model or **Tabular** model is loading data from a data mart and the processing is doing calculation of KPI and display a pivot in a data access BI tool. OLAP stands for OnLine Analytical Processing. Power BI

A **report** with criteria parameter do search, select and calculate data from a data mart based on an ad hoc query of a business user, or the report is send out as pdf file, or Excel file to the business users every day, week, month and so on.

A **dashboard** is a collection of unrelated data intended to summarize the state of the business during a given time frame.

A simple dashboard sample.

	This week	Amount	This month	Amount	YTD	Amount
New sales	12	$3,054	41	$12,469	478	$118,608
New customers	3		11		97	
New employees	0		2		6	

A dashboard provides a quick, clear and clear overview of the key figures, indicators (KPI) and performance data that are most important to him or her, shown as numbers and data visualization linked to business user groups. »If your dashboard does not fit on one page, you have a report, not a dashboard.« cf. Avinash Kaushik.

Start write a data story while designing a dashboard where the design should follow the narrative with a destination to answering the business questions. Dashboards for business needs to answer key business questions where every chart and graph and metrics and KPIs gives actionable insights and helping a stakeholder or a business user to make better data-driven decisions.

Percentage (fraction) and ratio based on a dimension slice will be calculated »on the fly« in a BI tool when a fact table contains a numerator and a denominator, e.g. a DAX (Data Analysis Expressions) query language for a tabular model in Power BI.
Power BI guidance to star-schema and best to use a star schema in Power BI.
DAX for SQL user and DAX tutorial.

Measure Average unit price as an intermediate calculation from a Sales fact.
Measure Active customers with a locale variable in a code block that is ending with a return in DAX from a Sales fact with a dimension key column Customer_key to count the number of different customers from a Kimball type 1 dimension Customer.

```
Average unit price := DIVIDE(Sum(UnitPrice x Quantity), Sum(Quantity), 0)
Active customers :=
  VAR measurevalue =
  CALCULATE(
    DISTINCTCOUNT('Fact_Sales'[Customer_key]);
      FILTER ('Fact_Sales';'Fact_Sales'[EndDate] = BLANK() &&
                'Fact_Sales'[SaleAmount] > 0))
RETURN IF (ISBLANK(measurevalue); 0; measurevalue)
```

Kimball calls this area a Business Intelligence Application.

Based on a fact table called Fact_Product with a business key in column Campaign_bkey and a business date in column LaunchDate, there can be carried out a date range lookup to a Kimball type 2 dimension called Dim_Campaign to fetch a value from dimension column CampaignName in DAX:

```
CampaignName := CALCULATE(VALUES('Dim_Campaign'[CampaignName]),FILTER(
'Dim_Campaign', 'Fact_Product'[Campaign_bkey]='Campaign'[Campaign_bkey] &&
                'Fact_Product'[LaunchDate]>='Campaign'[ValidFrom] &&
                'Fact_Product'[LaunchDate]<'Campaign'[ValidTo]))
```

10 simple steps for effective KPI. Data driven Infographic

This area is also called data consumption layer or data delivery layer for reporting, analysis and self-service BI.

1.7.8. Supporting databases

Supporting data in tables for a data warehouse can be placed in either several databases and/or in several database schemas in a database to be across the data areas. Here is some examples.

Audit with auditing data to an audit trail for reconciliation check from source systems with number of rows in tables and other summarized measures to monitor and control to avoid discrepancy between source systems and the data warehouse and make sure that all data have been fetched and saved in the areas of the data warehouse by doing validation check. Do alert the data owner and data users by email. Auditing is a kind of inspection (in danish eftersyn). Read more about data reconciliation in section 1.13.

System with data for a ETL process e.g. connection string, capture execution log metadata (at runtime) e.g. ExecutionLogId, execution time, status of execution, error message, job log, package, transform and exception handling.

To save a value of the »latest value« for incremental load or delta load with delta data detection in a **managing log**.

To save values of IdaBatchDataCaptureId, IdaBatchDataCaptureTime, ArcBatchDataCaptureId, ArcBatchDataCaptureTime.

Test with test cases while programming a ETL process and for making KPI values to validate the program.

Usage with custom data for mapping and consolidation of data from multiple source systems and data for dimension descriptive context, hierarchy, group, band, sort order and rules for new dimension data. Usage tells how source data is transformed and changed into useful information by a ETL process to meet the requirements of the business and displayed in a data mart. Usage data is maintained and updated by the business users through an application.

For example, product data can be found in multiple source systems with different spellings where a usage application will help the business to do the unite, interconnect, compile, conform and consolidate the products into one spelling for a product together with a mapping of the multiple source systems product numbers into one product.

The usage application can also provide hierarchy levels, group name, band intervals, sort order for the products. Some data in the Usage supporting database will be loaded from the archive, because the data comes from a source system, to be enriched by the business users. Input data area will fetch data from the Usage supporting database and further to the archive to be used in transformation and updating dimension with enriched data that doesn't exists in the source system. Usage data will also be used for making fact data e.g. a calculation with a constant value, a rule as if-then-else and a lookup value.

Key mapping tables can be part of a Usage database in their own database schema Keymap or by including Keymap in a key mapping table name. When there is no Usage database, a key mapping table will be part of a Data staging area.

Usage is data-driven to avoid data values inside a ETL package, the sql statements and the program. A Master Data Management (MDM) is an extension of a Usage database (in danish brugerdefinerede stamdata, ordet custom kan oversættes til specialbygget). Examples of Usage data in section 6.1.

Wrong with data that can't be loaded into a target because of error description e.g.: »The data value violates integrity constraints«, »Violation of primary key constraint«, »Cannot insert the value null«, »Cannot insert duplicate«, »String data would be truncated«, »Arithmetic overflow error«, or incorrect data or a data has outliers (deviations).

Wrong data is handled in a data staging area to filter wrong data and error data away for making correct data to next area. Wrong data is rejected data like when a ETL process is removing duplicate rows to keep them in a wrong table to be monitor by a data steward that can inform a source system. Wrong data can also be data that do not follow the quality assurance and quality control. Quality assuring, consistency and integrity is important parts of a ETL process and the audit trail.

Wrong tables can be part of a Usage database in their own database schema Wrong or by including Wrong in a wrong table name. When there is no Usage database, a wrong table will be part of a Data staging area.

I do not recommend that a data warehouse contain data according to the principle of Garbage in - Garbage out (GIGO), because wrong source dates e.g. 0205-02-07, 1899-09-25, 3987-11-22 or 9999-12-31 or 9999-99-99 does not exists in a Date dimension and can create an issue of the range of dates are not supported. Maybe an old date comes together with other wrong values, which can make a total KPI in a Power BI report looks suspicious for the business users. Is it better to move wrong to another location in a data warehouse and send an email to business users to have it corrected in the source system and later on the good data will be loaded.

Read more about data quality in section 1.14 and about an Audit dimension in section 4.4 to tag a wrong data row in a target table.

When custom data in the Usage database is going to be changed it can be done in a UAT user acceptance testing environment. After a ETL process has been executed we can test the data warehouse. Later we can take a backup of a Usage database and restore it in a production environment.

1.8. Data integration mapping by acronyms examples

Let's have a look at an example that is using a key mapping table to make a solid mapping for a dimension called SalesDevice in a data mart area (DMA).

A Usage database has a two key mapping tables for sales devices from two operational systems and each table has its own business key columns with a suffix _bkey (or BK).

A Usage database has a mapping table with a immutable surrogate business key column with a suffix _sbkey which contains acronyms as a simple letter **code** that can be used in a dimension in a data mart for a dropdown box search criteria for a report and for a dashboard, or for a business user ad hoc query in a tool like SQL Server Management Studio, DBeaver, Access, Python or RStudio.

A surrogate business key is called a retained key in SAS and a persistent durable supernatural key in Kimball. A surrogate business key column is mapped to a business key column in a key mapping table which make it stands for an integration and a merge to a conformed dimension and shared dimension.

A table SalesDevice_Map has a surrogate business key column with a suffix _sbkey as a primary key merging of all devices from retailer shops and from online shop to a current name and a sort order and a category or classification or group called DeviceModel. The table SalesDevice_Map will be updated by a data steward in a Usage database and a ETL process will flow the data to a Dim_SalesDevice dimension in a data mart area.

The example uses different database schemas: Usage for a usage area, dma for a data mart area, map for a mapping of data, source for a data source system and different labelling as prefix or suffix in table names and view names: Dim_, Fact_, _Map, _Keymap and _Lookup.

usage.SalesDevice_Map

SalesDevice_sbkey	SalesDevice	Sortorder	DeviceModel
CR	Cash register	10	Register
ECR	Electronic cash register	20	Register
TSCR	Touch screen cash register	30	Register
CM	Cashier machine	40	Machine
PC	Personal computer	50	Computer
MAC	Mac	60	Computer
IPD	iPad	70	Computer
IPN	iPhone	80	Phone
SM	Smart phone	90	Phone
TC	Tablet computer	100	Computer
MR	Manually register	110	Register

A retailer shop has different sales devices over time.

A key mapping table SalesDevice_Keymap_Retailer has a business key column with a suffix _bkey from different source systems mapped to the relevant surrogate

business key with a suffix _sbkey and timeline metadata columns ValidFrom and ValidTo to show an active period.

usage.SalesDevice_Keymap_Retailer

SalesDevice_bkey	SalesDevice_sbkey	ValidFrom	ValidTo
9001	CM	1900-01-01	9999-12-31
9000	CR	1900-01-01	9999-12-31
1	ECR	1900-01-01	9999-12-31
20	MAC	1900-01-01	9999-12-31
9002	MR	1900-01-01	9999-12-31
10	PC	1900-01-01	9999-12-31
2	TSCR	1900-01-01	9999-12-31
-2	-2	1900-01-01	9999-12-31
-1	-1	1900-01-01	9999-12-31

A online shop has different sales devices over time.

A key mapping table SalesDevice_Keymap_Online table has a business key column with a suffix _bkey from different source systems mapped to the relevant surrogate business key with a suffix _sbkey and timeline metadata columns ValidFrom and ValidTo to show an active period.

usage.SalesDevice_Keymap_Online

SalesDevice_bkey	SalesDevice_sbkey	ValidFrom	ValidTo
1	IPD	1900-01-01	9999-12-31
2	IPN	1900-01-01	9999-12-31
MAC	MAC	1900-01-01	2010-12-31
100	MAC	2011-01-01	9999-12-31
PC	PC	1900-01-01	2010-12-31
101	PC	2011-01-01	9999-12-31
20	SM	1900-01-01	9999-12-31
10	TC	1900-01-01	2015-12-31
10	PC	2016-01-01	9999-12-31
15	TC	2016-01-01	9999-12-31
-2	-2	1900-01-01	9999-12-31
-1	-1	1900-01-01	9999-12-31

We see that sales devices PC and Mac has different business key values in source systems compared to the retailer key mapping table, and over time sales devices have changed the business keys in a source system from a letter code to a number value. Above business key 10 represents two different sales devices, a Tablet computer (TC) to the end of 2015 and a Personal computer (PC) from 2016.

For example, above business key value 1 has two mappings, ECR for a retailer Electronic cash register and IPD for an online user iPad. A ETL process will for transaction sales data do a lookup in the relevant key mapping table to fetch the surrogate business key value and another lookup in the SalesDevice dimension to fetch the dimension key value to be saved within a fact row to a Sales fact table.

A dimension table Dim_SalesDevice has a dimension key column with a suffix _key as a primary key, and two rows for artifact values or inferred members for Missing and Unknown.

dma.Dim_SalesDevice

SalesDevice_key	SalesDevice_sbkey	Device-Model	SalesDevice	Sort-Order
-2	-2	None	Unknown	-2
-1	-1	None	Missing	-1
1	IPD	Computer	iPad	70
2	MAC	Computer	Mac	60
3	PC	Computer	Personal computer	50
4	TC	Computer	Tablet computer	100
5	CM	Machine	Cashier machine	40
6	IPN	Phone	iPhone	80
7	SM	Phone	Smart phone	90
8	CR	Register	Cash register	10
9	ECR	Register	Electronic cash register	20
10	MR	Register	Manually register	110
11	TSCR	Register	Touch screen cash register	30

The SalesDevice dimension has values or members as rows in a table in database.

An example of a view that binds a key mapping table and a dimension table together to be used in a ETL process, e.g. a sql statement:

```
CREATE VIEW keymap.SalesDevice_Retailer_Lookup AS
SELECT
  keymap.SalesDevice_bkey,
  keymap.ValidFrom,
  keymap.ValidTo,
  dim.SalesDevice_key
FROM usage.SalesDevice_Keymap_Retailer map
       INNER JOIN dma.Dim_SalesDevice dim
         ON dim.SalesDevice_sbkey = keymap.SalesDevice_sbkey
```

Retailer transaction sales data will join to the above view to let business key values fetches dimension key values and to insert rows into a Sales fact table.

It is called **Date Range Lookup** to fetch a dimension key value in a dimension table through a business key value and a transaction date value, and save a dimension key value in a fact table row. (In danish dato-spænd opslag)

For example dimension key SalesDevice_key from dimension dma.Dim_SalesDevice through a business key value SalesDevice_bkey, where both keys are combined in the above view together with timeline metadata columns ValidFrom and ValidTo. A retailer transaction sales data with a PurchaseDate must be between timeline metadata columns ValidFrom and ValidTo to fetch a dimension key value before insert a row into a Sales fact table dma.Fact_Sales.

In this example the data capture is **incremental load** for column TransactionId with first to fetch the greatest value from a fact table Fact_Sales, and then fetch source data table rows with values greater than it.

To avoid a null in a dimension key in a fact table (called »handling null foreign keys in fact tables«) two integer artifact values or inferred members are used: -1 refers to missing and -2 refers to unknown.

For example in a sql statement with two dimension key columns for Date and SalesDevice with suffix _key, and one business key column with suffix _bkey:

```
INSERT INTO dma.Fact_Sales
  (PurchaseDate_key, SalesDevice_key, SaleAmount, TransactionId)
SELECT
  trn.PurchaseDate, -- date format same as the Date dimension role-playing Purchase.
  SalesDevice_key = ISNULL(sdrl.SalesDevice_key,-2), -- in case of no match in join.
  trn.SaleAmount,
  trn.TransactionId
FROM source.RetailerTransactionSales trn
       LEFT OUTER JOIN keymap.SalesDevice_Retailer_Lookup sdrl
         ON sdrl.SalesDevice_bkey = ISNULL(trn.SalesDeviceNumber,-1) AND
             trn.PurchaseDate >= sdrl.ValidFrom AND trn.PurchaseDate < sdrl.ValidTo
WHERE trn.TransactionId > (SELECT MAX(TransactionId) FROM dma.Fact_Sales)
```

Please notice, that the dimension key PurchaseDate_key is of the data type date.

The above sql statement in a ETL process context:

- **E**xtracting as incremental load with a Where clause to fetch source data in table alias trn that has a greater value than the max value from the fact table.
- **T**ransforming of source data business key trn.SalesDeviceNumber to dimension key SalesDevice_key and to handle null as -1 for missing and not exists value as -2 for unknown.
- **L**oading new data rows into a Sales fact table with a SalesDevice_key as a reference to the SalesDevice dimension in a data mart area (in a dma schema).

Horoscope star signs as a static dimension example

A Usage database has a table with names of horoscopes as a natural key, an abbreviation as a business key labelled _bkey, and I am using the planet/house cusp number as a business surrogate key labelled _sbkey. Element becomes a group of the twelve signs. (icons)

Element	Horoscope	BeginAt	EndAt	_bkey	_sbkey
Air	Aquarius	January 20	February 18	AQ	11
Air	Gemini	May 21	June 20	GM	3
Air	Libra	September 23	October 22	LI	7
Earth	Capricorn	December 22	January 19	CP	10
Earth	Taurus	April 20	May 20	TA	2
Earth	Virgo	August 23	September 22	VI	6
Fire	Aries	March 21	April 19	AR	1
Fire	Leo	July 23	August 22	LE	5
Fire	Sagittarius	November 22	December 21	SG	9
Water	Cancer	June 21	July 22	CC	4
Water	Pisces	February 19	March 20	PS	12
Water	Scorpio	October 23	November 21	SC	8

1.9. Data capture or data ingestion with delta data detection

Data capture or data ingestion or ingesting or ingest of data from a source system to a Landing zone area and/or an Input data area is based of two data flow directions: (in danish datafangst, hjemtagelse, indtage, modtagelse)

Push where a Source system will find the data and deliver it to an area like a database or as a file to a FTP server in a Landing zone area. Later an Input data area will fetch the data and indicate that the data is downloaded, received and stored, so the source system knows that it can make a new delivery by using a semaphore to control access to a common resource by multiple processes. A snapshot on a source system to create a read-only and static view as a transactionally consistent.

Pull where a Landing zone area will fetch a file from a FTP server, or where a Landing zone area or an Input data area will fetch data from a Source system through a sql Select From Where statement maybe with joins of tables, or using a view or calling a stored procedure with a parameter as a criteria value to limit source data, and for doing delta data detection. From a ETL tool or a web-service where a data warehouse will request for source data.

Granularity of data capture integration strategy is to consider when the amount of data is big in a source system. A data warehouse prefer to receive at the lowest granularity level of detail in case of specific analysis usage or data validation, but sometimes it is necessary to aggregate and summarize source data to a higher granularity like per day and per customer segment for transaction data or transactional data from an OLTP system e.g. orders, invoices, billings, payments, site hits.

Non-transactional operational system data e.g. Customer, Location, Contact, Supplier, Part, Product can be stored in a **Master Data Management** (MDM) database as master tables that is shared with multiple operational systems in an organization to avoid data redundancy in the sense that the same data must be updated in several systems by a business user. Instead, master data is updated only in one place by a business user, after which data is automatically delivered to a number of operational systems to achieve one truth about data. The operational systems submit data to MDM and the operational systems subscribe to and continuously receive current data as needed, possibly including versioned historically collected data.
MDM is an extension of the Usage supporting database.

Data capture from a source system is based on multiple data delivery forms:

- Data rowset, recordset, multiple result sets from a relational database.
- csv file with a separator like comma or semicolon, maybe with field header.
- tsv file e.g. from Excel save as unicode text with tabular separator, txt file.
- XML Extensible Markup Language with XSD XML Schema Definition.
- JSON JavaScript Object Notation with JSON Schema.
- JSONL JSON Lines.
- TIP Nasdaq specification that looks like a mix of xml and csv formats l1, l2.
- Parquet from Apache or Delta Parquet with partition.

Data capture from a source system is based on three data delivery methods:

Full load or full dump as flush and fill where Landing zone area or Input data area gets all data from a source system, a full data set.

Incremental load where Landing zone area or Input data area gets new data from a source system and insert data into a target.

Needs an Initial load the first time running. It can be a source system that keep track of delivered date, or it can be the Input data area that keep track of a latest received Sequence Id (integer), Sequence Time (datetime), LastUpdatedDate, LastUpdate, LatestTS, LastModifiedDate, updated_at, ingested_at, created_at, creation_time or InsertUpdateTimestamp value from a source system for each data row, so an Input data area can detect new data by asking for data since that latest received value or a maximum value found in the last delivery, e.g.:
InsertUpdateTimestamp > the latest received value or last update datetime.

Some posting data has an entry date (or a created date):
EntryDate > »the latest received EntryDate«.

(In danish Posteringsdato eller Bogføringsdato).

Incremental load from an archive table with column ArcRecordId, e.g. partial sql:

```
FROM archive.Sales sal
WHERE sal.ArcRecordId > (SELECT MAX(ArcRecordId) FROM dma.Fact_Sales)
```

I prefer using a Id column for incremental load where Id is an artificial auto-generated unique sequence number or an identity column and it can be a clustered index as a primary key.

Incremental load is to fetch new data from a source and insert data into a target. In the sql example there is a source table Input.Posting and a temporarily target Staging.Accounting and a final target Fact.Accounting.

To avoid a null in a dimension key in a fact table (called »handling null foreign keys in fact tables«) two integer artifact values or inferred members are used: -1 refers to missing and -2 refers to unknown.

For example in a sql statement with three dimension key columns for Date, Customer and Product with suffix _key, and two business key columns with suffix _bkey:

```
TRUNCATE TABLE Staging.Accounting
INSERT INTO Staging.Accounting
        (EntryDate, CustomerId_bkey, ProductNumber_bkey, Amount)
SELECT EntryDate, ISNULL(CustomerId,-1), ISNULL(ProductNumber,-1), Amount
FROM Input.Posting p
WHERE p.EntryDate > (SELECT MAX(EntryDate) FROM Fact.Accounting)

INSERT INTO Fact.Accounting(EntryDate_key, Customer_key, Product_key, Amount)
SELECT
  stg.EntryDate, --date format same as the Date dimension role-playing Entry.
  Customer_key = ISNULL(dc.Customer_key,-2), -- in case of no match in join.
  Product_key    = ISNULL(dp.Product_key,-2),    -- in case of no match in join.
  stg.Amount
FROM Staging.Accounting stg
        LEFT OUTER JOIN dim.Customer dc
          ON dc.CustomerId_bkey = stg.CustomerId_bkey AND
              stg.EntryDate >= dc.ValidFrom AND stg.EntryDate < dc.ValidTo
        LEFT OUTER JOIN dim.Product dp
          ON dc.ProductNumber_bkey = stg.ProductNumber_bkey AND
              stg.EntryDate >= dp.ValidFrom AND stg.EntryDate < dp.ValidTo
```

Delta load where Landing zone area or Input data area gets new data or changed data or removed data from a source system, and using a delta data detection is to be able to insert new data, update changed data and delete removed data into a target. There is tools called merge and upsert.
Delta load is a complete net change of insert + update + delete.
For example, for sales with yesterday's new orders, cancelled orders and updated orders that can include all the updates over the whole day, where we could call it versions of orders. Price updates, etc, to make target matches source exactly.

Source data is fetched by incremental load and a delta data detection examines the source data to determine whether they are new data, changed data or removed data, and to do an operation at the target as insert, update, delete, where delete can be hard or soft with a metadata column IsDeleted_meta, and update can be hard or soft with a history by using a update and a insert.

Incremental load or delta load is very relevant for a **real-time data warehouse** with continuously updates throughout the day to be displayed in a dashboard and when the amount of data and number of rows are big and a full load takes longer and longer time, as time goes by.

The data warehouse can do its own delta data detection between Input data area and Archive area to identify new data, changed data and removed data to maintain historical data with datetimestamps to handle following actions:

- Which rows is new, give them a NewDateTime, CreatedDateTime, InsertTime and a flag N for New data.
- Which rows has been changed, modified, altered, updated, give them a ChangedDateTime, ModifiedDateTime, Latest ChangedTime, UpdateTime and a flag C for Changed.
- Which rows has been removed give them a RemovedDateTime, DeleteTime and a flag R for Removed.
- Which rows has been corrected e.g. correction of a mistyping, a typos or a misspelling of a customer name, give them a CorrectionTime and a flag X.
- Which rows has been cancelled e.g. a wrong customer, give them a CancelledTime and a flag L for canceLled.
- Which rows has been terminated e.g. a product not for sale anymore, give them a TerminatedDateTime and a flag T for Terminated.

Sometimes a source system provides flag information to pass a status to a data warehouse. I have seen combination of consistent statuses like these:

- New – Changed – Removed

- New – Changed – Deleted
- Created – Modified – Deleted or Destroyed
- Create – Read – Update – Delete (CRUD)
- Inserted – Updated – Deleted
- Searching – Creation – Reading – Updating – Deleting (SCRUD)
- Added – Changed – Removed

Microsoft Windows calls it Add/Remove Programs in Control Panel (ARP).

To insert new data, to update changed data and to delete removed data from a source system

Change Data Capture CDC is a method to log data in a table when data is changed by transfer a copy of a data row from the table (or table log) to another table in a Landing zone area and/or an Input data area.

Changed data represents: New data, Changed data, Removed data or Inserted data, Updated data and Deleted data. Fetch the data values that has changed since latest fetching data (extract).
(In danish Data ændringsdetektion d.v.s at opsnappe eller at spore at data har ændret sig i et andet system).

The CDC implementation of Microsoft SQL Server allows the following types of operation on the data: Delete, Insert, Update (old values), Update (new values).

Sometimes a source system update a data row multiple times but only the most recent version goes to the data warehouse. If a source system insert and update a row before a new load to data warehouse, only the updated version goes to data warehouse. If a source system insert a new row and delete the same row before a new load to data warehouse, the data will never goes to data warehouse. It is seldome that a source system has a log and is logging all changes with a revision number or a datetimestamp, therefore a data warehouse does not contains 100% full historical updates of data in an organization. Sometimes a source system has for each (or selected) tables in the database an extra historical log table that contains all inserted, updated and deleted data rows from the original table together with an Action column (values for Insert, Update, Delete) and an EntryDate datetimestamp column that mark when the action was occurred and a SequenceId that is used for incremental load to pull data from a Source log to an Input data area, maybe through a Landing zone area.

An example of a source historical log table with rows every time a customer change data. At date 2005-12-01 a customer with maiden name Marie Beaulieur makes her first purchase in a store and gets her membership registered with an CustomerId 421 in an operational system. She is inserted into log table that keeps CustomerId as a business key, and that add a unique sequence id as primary key for the log for having independence of the business key.
Marie Beaulieur got married at 2007-11-16 and took her new husband's surname, therefore her name is changed to Marie Lesauvage, and she is inserted into log table with a new row.
Marie Lesauvage remarried at 2014-07-15 and took her new husband's surname, therefore her name is changed to Marie Sainte, and she is inserted into log table with a new row.
At date 2017-02-07 she stopped her membership and was deleted as a customer in the operational system, and she is inserted into log table with a new row.
Marie's wedding dates and dates for start and stop membership is in general called business date or date of change, and in a log table as column LogEntryDateTime.

SequenceId	LogEntryDateTime	LogEntryAction	CustomerId	CustomerName
626	2005-12-01 10:12:34.4906240	Inserted	421	Marie Beaulieur
7503	2007-11-16 14:37:07.2353231	Updated	421	Marie Lesauvage
19473	2014-07-15 11:09:33.7629562	Updated	421	Marie Sainte
25003	2017-02-07 23:57:21.9876543	Deleted	421	Marie Sainte

Implemented as after-insert/update/delete triggers in a source system.

Another example, where Jane Macquarie was sales manager for the eastern region until December 31, 2018, and then took responsibility for the western region from January 1, 2019. How was it registered in a source system? I have seen systems without a date to register a shift, because an operator will update the responsibility at beginning of January 2019, which means the data warehouse can let Jane get a timeline metadata column ValidFrom date as 2019-01-05 for western region, but it is a wrong picture and sales fact data for few days in 2019 will belong to eastern region. Please, always ask a source system for multiple date columns, e.g. business date or date of change, and use them for timeline metadata columns ValidFrom and ValidTo to get a real picture of the business in archive area and data mart area.

Read more in Kimball Design Tip #63 and Kimball ETL Toolkit page 188-189.

Data latency describes how quickly source data has to be ready in the data warehouse for the business users to do their reporting. Deadline is normally in the morning based on a periodic batch of source data from yesterday. For a real-time load to a data warehouse with constantly new presentation in a dashboard, a ETL process must be streaming oriented where source data continuously flows into the data warehouse by incremental load or delta load to do transformation and making conforming data.

1.10. Enterprise data warehouse modeling

A data warehouse can be modeled in different ways. I will present three data models. In a data warehouse solution the source data from multiple source systems can go into a single common database called an Enterprise data warehouse (EDW) where data is integrated into a single and coherent structure. Notice, to model a EDW it is based on business needs and not based on source data structure.

A EDW contains and represents a »**single version of truth**« for the enterprise.

1.10.1. Inmon modeling

Bill Inmon is using a Operational Data Store (ODS), a place to store source data as a central database that provides a snapshot of the latest source data from multiple source systems (operational systems or transactional systems) with validate referential integrity for operational reporting and as a source of data to an Enterprise data warehouse (EDW) which feed the Corporate Information Factory (CIF), which provides a logical framework for delivering business intelligence.

When a EDW is modeled after Inmon modeling to a Inmon model it is based on Peter Chen's Entity Relationship modeling with super-sub entity, associative entity and 80% normalized data often at third normal form (3NF) as in a relational model.

The EDW offers integrated, granular, historical and stable data that has not yet been modified for a concrete usage and can therefore be seen as neutral. It can keep historical data meaning all the changes to the data or only the days-end status e.g. End Of The Day for each data revision from ODS (or Archive area or Input data area).

An entity Person can have a one-to-many relationship to the addresses of the person's living places in another PersonAddress entity with timeline metadata columns ValidFrom and ValidTo, and these entities or tables can be merged into a type 2 dimension table in a Kimball data mart.

The EDW modeling is subject-oriented meaning all business processes for each subject e.g. customer needs to be modeled. A EDW common data model is not technical, it must be based of business processes and all source data is integrated and collected into one common database.

Inmon modeling can divide or partition data into super-sub tables to place related columns in its own sub table to separate from other columns. There is often a one-to-one relationship between a super table and its sub tables. We can say that this kind of division of data is subject-oriented based. A super table can contain a business key (natural key, surrogate key) from a source system together with other basic data columns, or instead we place the business key in a mapping table connected to each super table where a translation of a business key becomes a data warehouse surrogate key Id as an integer or a guid, and we let the Id become primary key in the super table and foreign key in the sub tables.

When a source system adds new data, the data warehouse can place data in a new sub table and connect data to the existing super table. Similarly, when a new source system is introduced to the data warehouse. Inmon modeling using super-sub tables allows for agile modeling, read more in section 1.10.5. Also called super-sub-entity or supertype/subtypes has two rules:

- Overlapping or inclusive e.g. a person may be an employee or a customer or both.
- Non-overlapping, mutually exclusive or disjoint e.g. a patient can either be outpatient or resident patient, but not both at the same time.

Let the design of the data model for a EDW start doing a »helicopter view« for identification of entities, e.g. salesperson or engineer will be stored in an Employee entity with a one-to-many to a Jobfunction entity, or maybe do more abstraction in a bigger picture making a Person entity where employees and customers users can be stored together. Later in the design process move on to a »weeds view« for all the details for columns, conformed names and data types.

In the EDW keep the source data with time span columns like Begin date and End date or Start date and Stop date, and sometimes also convert the source data to row wise entity as a kind of transactional data that is a sequence of information exchange like financial, logistical or work-related data, involving everything from a purchase order that becomes an invoice with multiple invoice line items, to shipped status, to employee hours worked, plan and activity records, for subscription period and to insurance costs and claims.

From Inmon model to Dimensional model
A EDW can be a data source to one or multiple data marts using Dimensional modeling with denormalized data controlled by a ETL process. A data mart can use a different modeling that fit better for use of data. A EDW should be accountable and auditable which by default means pushing business rules of changing/altering data downstream to »between the data warehouse and the data marts«.

EDW tables and views can be data source for a **tabular model** with a DAX formula to do filter and calculate in-memory KPI to be visual in Power BI, where a data mart area is skipped or been replaced by EDW views, therefore it is called a ELT process because the transformations is executed on the way to a business user.
W H Inmon history look

1.10.2. Anchor modeling
When a EDW is modeled after Anchor modeling to a Anchor model it is based on four elements that holds on historical and time-variant raw data in entities/tables with labels:

Anchor is a surrogate key generator for a source system business key from a data warehouse point of view.
Attribute has context descriptive data values from a source system connected to the business key therefore attribute has a foreign key reference back to the anchor. The attribute table takes care of history, therefore it has also a surrogate key.
Knot is a lookup table with basic data.
Tie is a relationship table between anchor tables to handle one-to-one, one-to-many and many-to-many relationships, therefore no foreign key in anchor or attribute table except to a knot table (»tie the knot« means getting married).

Anchor modeling has only extension of new tables and none modification of existing tables. This ensures that existing data warehouse applications will remain unaffected by the evolution of the data warehouse. When a business key exists in multiple source systems, there will be one common anchor table and several attribute tables, because each source system has its own attribute table.

Anchor modeling allows to build a real-time ETL process to a data warehouse where some attribute tables needs a more frequent update than others to provide fresh data to the business users in a report or for a online dashboard. Other attribute tables only needs updating per hour or at end-of-day.

From Anchor model to Dimensional model
For a ETL process from a Anchor model to a Dimensional model, think that an Anchor and its Attributes become a Dimension, and a Tie and its Attributes become a Fact. A view will mix anchor and attribute, where surrogate key of anchor becomes a durable key, and surrogate key of attribute becomes a dimension key. Both keys will be added to a fact as a Kimball type 7 dimension, and the fact can join to a view to fetch the current values, and join to a view to fetch the registered values when the fact data occurred with its transaction date. A tabular model in Power BI can choose the content for its presentation interface because all data are on a silver platter.

Anchor modeling diagram example [Olle Regardt and Lars Rönnbäck] Read more.

1.10.3. Data vault modeling
When a EDW is modeled after Data vault modeling to a Data vault model it is based on component parts of a Core Business Concept (CBC) as an ensemble consisting of three components that holds on historical and time-variant raw data in entities or tables with labels:

Hub is used to store a business key.
Link is for a relationship between hubs.
Satellite has context descriptive data values from a source system.

They all have a LoadDate column (Load_dts, LoadDateTime, ValidFrom) to show when a data row was entered.

Each component can be drawn with its own color.
Let me elaborate with some examples.

Hub (blue) separates the business keys from the rest of the model and will translate business key to a unique hashbyte key value. A composite business key of

multiple columns will also become a hashbyte key column. E.g. a HubProduct has ProductHashKey together with business key ProductNumber. A hub is an integration point of a business key or a unique identifier and will never change. A hub exists together with at least one satellite. A data vault table has a **RecordSource** column as a reference back to a source system for data lineage e.g. value "Dynamics365.Sales.Product" and can be a multi-valued column with multiple sources. (Hub in danish er et samlingspunkt, knudepunkt for én forretningsnøgle som har data fordelt over flere satellitter der kredser om en hub eller et link).

Satellite (yellow) [sat] stores the context, descriptive data values and measure values in columns of either a hub or a link. A satellite is connected to one hub or one link, and a hub or a link can have multiple satellites. E.g. a SatProduct with foreign key ProductHashKey from HubProduct and data values for Name, Category, Target group. Satellites has all the relevant data for the data warehouse. Satellite is a history-tracking to handle historical data values for changes (updated/deleted) at any time in a satellite, where the primary key is composite of HashKey+LoadDate. A hash difference HashDiff column is a checksum of all data value columns for making an easy comparison for Kimball type 2 dimension.

To be 100% insert only compliant there is no columns for EndDateTime, ValidTo and IsCurrent flag, therefore no updates of rows in a satellite. If a source system can tell that a product has been expired or deleted then a new row is inserted into SatProduct with a current date value in column DeletedDate.

One hub or one link will usually have several satellites associated because we will regroup data into multiple satellites by classifications and types of data and information and by rate of change, so each of them can have its own granularity and timeline. Split logical groups of data into multiple satellites. With multiple source systems the RecordSource column is helpful when data sets is similar, else let each source system has its own satellite including a **RecordSource** column and share the HashKey from the hub to all its satellites.

Employee satellites example
We start with an employee hub with a business key column EmployeeNumber that is translated to a column EmployeeHashKey that will be in all satellites sat tables.

The logical model terms Hub and Sat become prefixes in the table names, and we should never apply physical design constraints to the logical model. An alternative could be database schemas, e.g. Hub, Satellite and Link for the tables.

HubEmployee(EmployeeHashKey, EmployeeNumber)
SatEmployeeBasic(SSN, BirthDate, MaidenName) constants/correction overwrites.
SatEmployeeName(FirstName, MiddleName, LastName, Gender).
SatEmployeeContact(CellPhoneNumber, Email, Skype, Facetime, LinkedIn).
SatEmployeeAddress with columns for address or a reference to an Address table.
SatEmployeeJob(FromDate, ToDate, Department, Title).
SatEmployeeHoliday(Date, Occasion).
SatEmployeeSalary(DateOfMonth, MonthlySalaryAmount, MonthlyPensionAmount).
SatEmployeeChristmasGift(Year, NameOfGift, AmountOfGift, LevelOfSatisfaction).

The above data comes from multiple source systems and data is added independently of each other to the individual satellites (divide and conquer). There will be outer join from a hub to its satellites and Inner Join from a satellite to its hub.

More satellites in addition to the above regular or standard satellite
Other satellites are shortly:

- Overloaded satellite provides the ability to combine data from multiple source systems into one satellite. Normally we have a satellite for each data source system.
- Multi-active satellite provides the ability to have multiple active rows for the same hub or link hashkey (business key), to handle a one-to-many relationship.
- Effectivity satellite provides the ability to track when a link is active with a time period or time span of begin date and end date, e.g. a source system contains current relationship of which buildings house which departments are located. Over time a department will relocate to another building. Effectivity satellite has no descriptive data columns.
- Record tracking satellite provides the ability to track when business keys were last seen provided by the source system in order to identify the difference between »missing for a few days« and »was deleted«, e.g. a business rule that all rows that have not appeared for 7 consecutive days should be treated as deletes.
- Status tracking satellite provides the ability to track the validity of rows in either hub or link (same-as effectivity satellite), e.g. audit trails or data from a change data capture (CDC) system to track information on CRUD operations within a source system.
- Computed satellite provides the ability to contain conformed and calculated data as a result of aggregation, summarization, correction, evaluation and it

might be the outcome of data quality routines, cleansing routines or address correction routines.

Link (green) integrate and capture relationship between hubs and links. E.g. a product is placed in a store, therefore we have a HubProduct and a HubStore with data values in a SatProduct and a SatStore, and a LinkProductStore represents the placement by combining ProductHashKey and StoreHashKey as foreign keys from the hubs to capture in which stores a product is placed and for sale. A link creates its own hashkey as a unique combination of the involved hub business keys e.g. ProductStoreHashKey from ProductNumber and StoreCode. Data values connected to a link is stored in its own satellite, e.g. SatProductStore with primary key/foreign key ProductStoreHashKey and data values for PlacementInStore and QuantityInStore. When a product no longer exists in a store, a ETL process will insert a new row in SatProductStore with the expiration date in the column DeletedDate.

A link handles one-to-one, one-to-many and many-to-many relationships because data vault has only optional many-to-many. A link is a dataless and timeless connection among hub business keys and is not a data-bearing relationship from ERD. A satellite on a link represents the history of the connection or the relationship.

Product Supplier Category link example
A Product table with foreign keys SupplierID and CategoryID will often be put together into a LinkProductSupplierCategory with ProductHashKey, SupplierHashKey and CategoryHashKey from HubProduct, HubSupplier and HubCategory and a primary key ProductSupplierCategoryHashKey. Data values connected to the link is stored in SatProductSupplierCategory. Over time a product can change supplier and category which is being handled in the SatProductSupplierCategory with a new row inserted with filled DeletedDate and a new row is inserted into link and sat for the new data combination with a new LoadDate. If the business one day allows a product to have multiple suppliers then the link is already prepared for it. In the future, the business will like to place products in stocks all over the world, and it will be registered in a new inventory management system, that creates a HubStock and a SatStock, and creates a LinkProductStock and a SatProductStock to handle all the new data and relationships.

Marriage example
A unary/recursive relationship (self join) e.g. LinkMarriage represents a connection between two persons with Person1HashKey and Person2HashKey from HubPerson with business key SSN and it gives primary key MarriageHashKey. All data is stored in SatMarriage(MarriageHashKey, MarriedDate, DivorcedDate, WidowedDate). A validation rule in SatMarriage ensures that one person can only be married to one person at a time else it will be marked as an error for an audit trail to catch. When a person enters a marriage and changes surname, it becomes a new row in SatPerson. When a marriage ends either by divorce💔 or by death✝ then a new row is inserted into SatMarriage with filled DivorcedDate or WidowedDate. Couples who are divorced and later remarry each other will reuse the same MarriageHashKey value from LinkMarriage for the new row inserted into SatMarriage. The LoadDate will be in order of the events. The DeletedDate will be used in connection with an error detection in the source system or an annulment of the marriage.

Another modeling of marriage: SatMarriage(MarriageHashKey, Date, MaritalStatus) with values of marital status: Married, Divorced and Widowed, makes it easy to extend with extra values e.g. Separated, Registered partnership and Abolition of registered partnership. When persons as partners are living together we will modeling it in a LinkCohabitation and a SatCohabitation. When persons only dates we make a LinkRelationship and a SatRelationship. Therefore each link and satellite is a flexible and agile way to divide data. (Agile in danish som smidig for forandringer, let og hurtig at udvide). In a generic perspective »helicopter view« we could have started with a general link to represent a connection between two persons with Person1HashKey and Person2HashKey from HubPerson, and label the link a general name like LinkCouple with a CoupleHashKey, and let the types of data create multiple satellites for the link as SatRelationship, SatCohabitation and SatMarriage. It is normal for a couple to start dating with data in the SatRelationship, living together with data in the SatCohabitation. When the couple gets married, the cohabitation timeline stops in the SatCohabitation, and the data recording continues in the SatMarriage. It is agile to think and data modeling in a general way and divide data.

Invoice example
An invoice becomes a HubInvoice where business key InvoiceNumber becomes InvoiceHashKey used in a SatInvoice with data values IssueDate, DueDate, DeliveryDate, PaymentMethod and PaymentDate. When a payment happens e.g. a week later, it will become a new row in the SatInvoice, but I prefer to add the payment to a new satellite called SatInvoicepayment with InvoiceHashKey, PaymentDate, PaymentMethod and PaymentAmount to divide data in separated satellites because of the time difference in data capture, and it allows for payment in installments in the future. If one day the sales department replaces the billing system then create a new SatInvoice. An invoice involves a customer in a store,

therefore a LinkInvoiceCustomer includes InvoiceHashKey, CustomerHashKey and StoreHashKey and combines business keys to InvoiceCustomerHashKey used in a SatInvoiceCustomer with data values like Reference no, Requisition number and Voucher number. An invoice line item involves a product, therefore a LinkInvoice-Product includes InvoiceHashKey and ProductHashKey and combines business keys to InvoiceProductHashKey used in a SatInvoiceProduct with data values LineItem-Number, Quantity, CouponCode, Discount (%), UnitPrice, VAT and SaleAmount. If the quantity in an existing invoice is corrected then a new row is inserted into Sat-InvoiceProduct. If a product is replaced then for the old product a new row is inserted into SatInvoiceProduct with filled DeletedDate and the new product is inserted into LinkInvoiceProduct and into SatInvoiceProduct.

More links
A link does not have its own unique identifier from a source system. Other links are: Nondescriptive link, Computed aggregate link, Exploration link and Link-to-link or Link-on-link.

Address example
Address will not be a Hub since the address itself is not a CBC but a description of a CBC e.g. Customer, Employee or Store. A CBC could be a CustomerDeliveryLocation as a hub with a satellite and a link to Customer hub because a customer could have more than one active location where goods can be delivered. The business key for the hub is using a concatenation of zip code, street name, house number to get a unique identifier for the CustomerDeliveryLocationHashKey.

Business ontology (ultra short)
Hub tables for business entities
Link tables for business processes
Satellites tables for data states
Read more. Ontology vs Data Model.

More component parts
There is several other types of component parts (entity/table), for example:

Transactional-link (light green) integrate and capture relationship between hubs and links to capture multiple transactions that involve the same business keys e.g. many sales to same customer of same product from same store, meaning multiple rows with the same set of keys. A transactional-link has its own unique identifier for a transaction (TransactionId, EventId) from a source system. Notice, that a trans-actional-link keeps its business key, it is not stored in an associated hub. If a revision of a transaction is received from a source system or a counterpart then insert the data values into the associated satellite. Since DV raw data is a structured archive, it will be wrong to calculate a counterpart row, but when data is extract-transform-load to a Transactional fact it is fine to calculate a counterpart for the fact.

Nonhistorized-link is for immutable data that has no edit history, in other words, constant data that will never be changed or deleted in a source system. Each transaction (data row) has a unique identifier (TransactionId) and there is no revision of a transaction in the source system. When there is a counterpart to a transaction it will have its own unique identifier and a reference identifier back to the original transaction.

Reference (gray) is referenced from a satellite and is the same as a lookup table with basic data e.g. RefDate or RefCalendar with many columns like Week, Month, Quarter and Year, RefMaritalStatus and RefMarriageType with Code and Text, and RefZipCode with Zipcode, City, State, State abbreviation for US. Date table is called a nonhistory reference table. A satellite is normalization to 3NF and contains only non-foreign key columns except simple reference values. If a reference (cross-reference or lookup table) contains history then move the old data values to a History-based reference satellite with primary key Code+LoadDate and Text con-tains the previous texts. I prefer to have a RefAddress with extra columns like State, Region, Municipality and Latitude and Longitude (gps coordinate), Kimball page 235.

Same-as-link (turquoise) [sal] indicates that two or multiple business keys for a hub are the same thing as a logical representation of a link where the business keys become rows together with the same hub hashkey value to map different business keys from multiple source system where each source has its own satellite. In a sal we can »merge together« differently named business keys to a single mas-ter key that is really the same-as the other business keys. Same-as-link is connec-ted to a hub and is used to identify when the same business object is identified by multiple business key values. Same-as link is a type of a Link to a Hub to handle different business keys and values within the same Hub. An example:

- HubCustomer(H_CustomerHashKey, CustomerBusinessKey) has a unique hashbyte key value for each business key value from different source systems with different column names such as CustomerId and CustomerNo.

· SalCustomer(S_CustomerHashKey, H_CustomerHashKey1,H_CustomerHashKey2) refers to HubCustomer twice to link two or several customer business keys together. The effective date of the business key relationship should be tracked in a Satellite connected to the Same-as-link.

Hierarchical-link (silverblue) [hal] to capture hierarchical relationship within a hierarchy, e.g. a product category hierarchy, between the business concept rows in same hub e.g. parent-child hierarchy. The effective date of the hierarchical relationship should be tracked in a Satellite connected to the Hierarchical-link.

Point-in-time (purple) [pit] is a helper table to a hub or to a link and is calculated by a ETL process based on all the satellites of the same hub or of the same link, because these satellites share the same HashKey together with different values in LoadDate. A pit table has its own LoadDate column (SnapshotDate), a column for the shared HashKey and columns for each involved satellite's LoadDate, therefore all the values of LoadDate from the satellites are represented in a pit table to be used to derive all valid versions for ease of joins of satellites and to improve the performance and includes to find the newest or most recent rows and to calculate EndDate or ValidTo. (A different implementation of a pit table is to have a row for each date in the calendar or for each working date in column LoadDate, but of course with 1000 rows in one satellite over a period of ten years it will give millions of rows but very easy to lookup on a given reporting date point-in-time as a transaction date). By adding a surrogate key sequence number to a pit table it can through a view create a virtual Kimball type 2 dimension. A pit table is extremely performant when it comes to getting data out of the data vault model. pit create and pit use.

Bridge (taupe) combines (or bridge) hashkeys and optional business keys from multiple links and their related hubs. A bridge can represent an aggregate level and can include derived and calculated columns.

Parallel loading
Does hubs and links need to ensure uniqueness constraints so it is impossible to load more than one source data table that hits the same hub/link at the same time? Data vault 2.0 accepts a low number of duplicate rows in a hub or a link because these duplicates have the same hashbyte key value and business key value, and will filter them with distinct on the way to a dimension and a fact, or remove them occasionally after the data has been loaded. Therefore data vault 2.0 is scalable because these solutions can improve the scalability and be parallelizable with parallelized loads.

HashKey - hashbyte key value - hashbyte value - hash value
Data vault 2.0 recommends for a Hub to hash a business key value instead of using an identity column as a unique sequence number because:

- HASHBYTES('SHA2_256', CONCAT(SSN, '**;**', UPPER(TRIM(Area)))) gives data type binary(32)/char(64). SHA1 use binary(20) and MD5 has been deprecated.
- Before hashing a business key or a data value or a text value we do some data operations as upper case, left and right alignment or trimming and save it in a column called »comparison business key« together with the original value in a Hub table. The hashbyte of 'edu' and 'EDU' is different therefore important to call a **upper** function before getting the hashbyte value.
- For Hash diff to make an easy comparison for data columns, we do the same data operations when it is agreed with the business and the data profiling.
- Hashing two values could create same hashkey value which is called a collision and is very unlikely but can't be left to chance or avoidance. If a data batch contains two different business key values which give same HashKey then bulk insert will result in primary key violation in a Hub. If a new business key value gives the same HashKey as an already existing one in a Hub then the loading can assume the new business key value is already there. Therefore first to check in Hub if new business key value exists and the HashKey exists then increase HashKey until no collision or using a collision sequence number, but there is no guarantee of same hash key values when data is reloaded. Maybe a need for a hash key detector detection job and detecting an undefined hash key. MD5 collision example. What to do when business key value is null or is an empty string?
- It is ease of copying data from environment to another like dev to test to prod without worrying about sequences being out of sync.
- When two source systems share same business key but in one source it is an integer data type and in the other source it is a string, by hashing they become same data type-in the hub table and sat table.
- When a source system is replaced, and the new source system comes with different data types e.g. from an integer business key to a string business key, it will still be hashed to same data type-in the data vault, therefore no maintenance.
- A composite business key of multiple columns become one hashkey column which is easy to use for join and merge.

- Easy to create a hashkey of a business key value when needed by a ETL tool or in a sql select statement instead of doing a lookup to find the sequence number stored in a table. But of course still need to make sure that the same business key is not inserted twice in same hub table and avoid collision.
- Do not use a HashKey column for a clustered index for the primary key in a hub for faster query join performance, because hashbyte has a non-sequential nature and therefore inserting new rows will take longer time. A unique non-clustered index should be considered for HashKey and a unique (non)-clustered index on the business key in a hub depending of the values. I have seen a hub with a unique clustered index on LoadDate+HashKey because first part of the value is ever-increasing. Read more.
- Loading to tables can be done parallelly e.g. at same time to a hub table and to a satellite table from the same source system table (parallel loading, enables parallelization). It sounds awesome! What if a ProductNumber is used in a Store relationship and will be inserted into the LinkProductStore with ProductHashKey and StoreHashKey but the ProductNumber is an orphane and does not exists in the Product table from a source system and therefore is not in the HubProduct then it gives a referential integrity violation in the data vault. Read about artifact values or inferred members in section 4.5.
- Business Key Collision Codes (BKCC).

Fetching data
Querying a data vault model implemented as tables in a relational database requires many more joins than Inmon model. Be aware it can be a performance issue to have columns of data type binary(32) or char(64) for storing millions of rows and for join many tables together compared by using an integer data type. Joining large pit, sat, link and hub together is expensive for performance and it is often debated by Linstedt. As far as performance goes, normalised structures are no where near as performant as a dimensional model. Nothing beats dimensional modeling for performance of queries.

Raw data vault area and Business data vault area
Data vault architecture has two data areas:

- Raw data vault area which I find is an archive of structure after a decomposition of a business entity / source data with almost no transformation to make a new composition to a data vault model where a business entity becomes hubs, satellites and links. An archive area or a persistent staging area that is isolated from business rules, which means no business rules transformations in a ETL process.
- Business data vault area called Business Vault after business rules transformations in a ETL process that contains data integration from multiple satellites to one satellite, business rules, business calculations and present data at different level of granularity. Pit and Bridge is part of a Business Vault to provide table to give a better query performance. Business Vaults can take many forms and it is about provisioning data for business users.

Linstedt said: »The Data Vault is for back-end data warehousing, not for production release to business users directly.«

Data vault allows to stay close to the source in terms of its granular objects. Data vault is flexible and easier to add new sources, more audit able and keep all data all the time so you will be able to always recreate a data mart model and do reload. Data vault has a data model that is auditable and scalable. It helps with flexibility on source specifications.

From Data vault model to Dimensional model
For a ETL process from a Data vault model to a Dimensional model, think that a Hub and its Satellites become a Dimension, and a Link and its Satellites become a Fact. Pit and Bridge is helping too. A view will mix Hub and Satellites, where Hub hashbyte key column becomes a durable key, and Satellite hashbyte key column becomes a dimension key. Both keys will be added to a fact as a Kimball type 7 dimension, and the fact can join to a view to fetch the current values, and join to a view to fetch the registered values when the fact data occurred with its transaction date. A tabular model in Power BI can choose the content for its presentation interface because all data are on a silver platter. I have seen dimensions and facts been implemented as views on top of a Data vault model, but be aware of performance when the amount of data increases.

When to use Data Vault? When a data warehouse has many source systems!

Order line example in Relational model 3NF, Dimensional model, Data Vault model

Data vault modeling diagram example [Dan Linstedt] Read more, extra more, more about loading, and there is many sites where individual persons give their recommendations. An example in a six minutes video and a longer video.
What if my business keys are not unique across the enterprise?.
Dan Linstedt comments my post at LinkedIn that he is using following colors:

green for hub, blue for link and white for satellite.

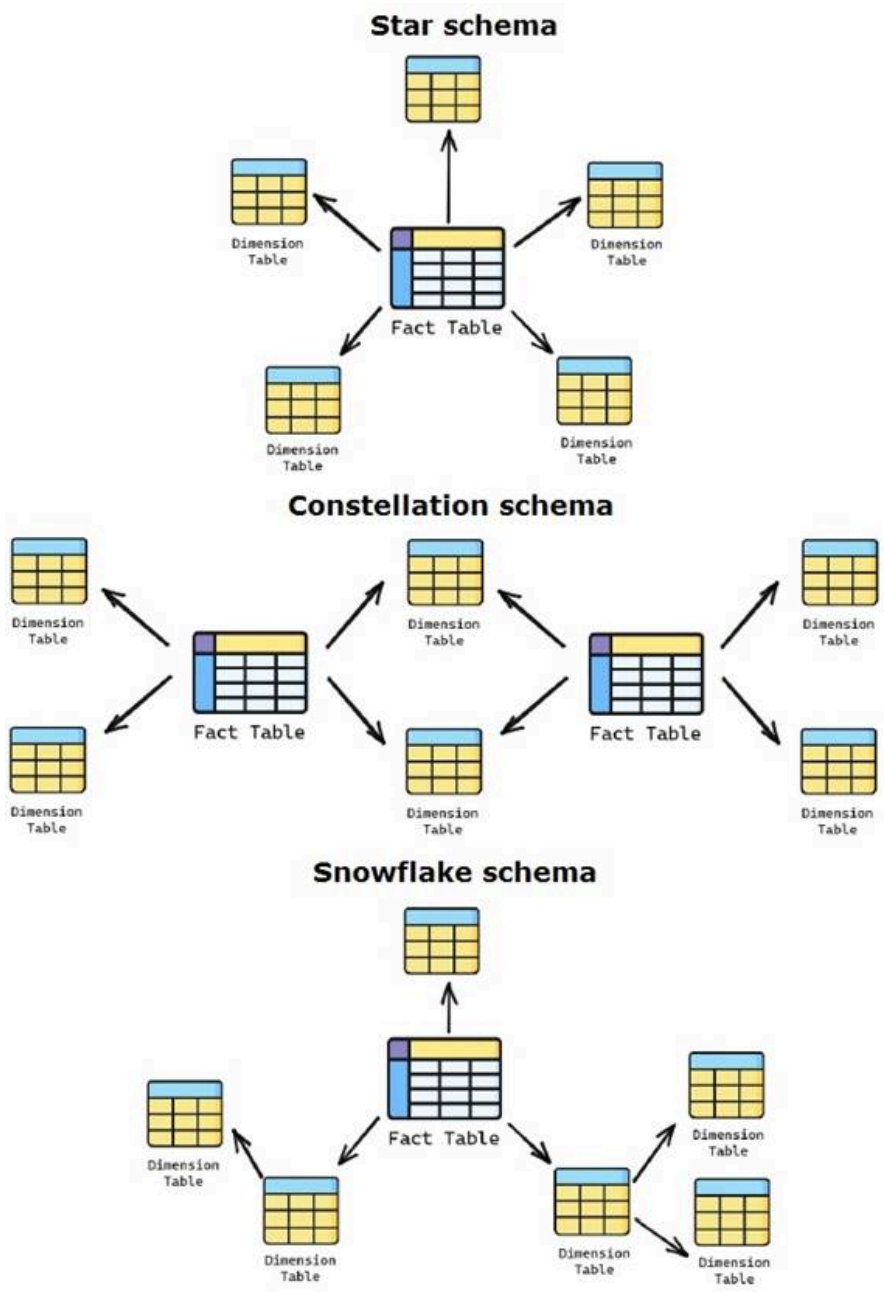
1.10.4. Dimensional modeling

Ralph Kimball does not like to store data in a EDW, he only store data in data marts that is using Dimensional modeling, therefore **EDW becomes a union of all data marts.**
We could say, we store data in an Enterprise dimensional data warehouse (EDDW).

Dimensional modeling ends up with a dimensional model as a:

- Star schema with one fact surrounded by conformed and shared dimensions.
- Constellation schema with few facts sharing conformed dimensions.
- Snowflake schema with one or few dimensions have become snowflaking.

A fact with measures, metrics, analysis variables is surrounded by conformed and shared dimensions with context descriptive data, text, hierarchy, group, band to explain the measurements and measures.



Some data modelers say, that Snowflake schema is when all dimensions have become snowflaking, but I say, a few dimensions which is called Starflake schema as a combination of star schema and snowflake schema: »Starflake schema aims to leverage the benefits of both star schema and snowflake schema. The hierarchies of star schema is denormalized, while the hierarchies of snowflake schema is normalized. Starflake schema is normalized to remove any redundancies in the dimen-

sions. To normalize the schema, the shared dimensional hierarchies are placed in outriggers.« source

I'm not a fan of the terms Snowflake schema and Starflake schema, because snowflaking is for a dimension only. Read about snowflake dimension and outrigger dimension in section 4.4. types of dimension tables.

Same goes with a centipede schema referring to a star schema with an excessive number of dimensions, resembling a centipede with a central fact table as its body and numerous dimension tables as its legs.

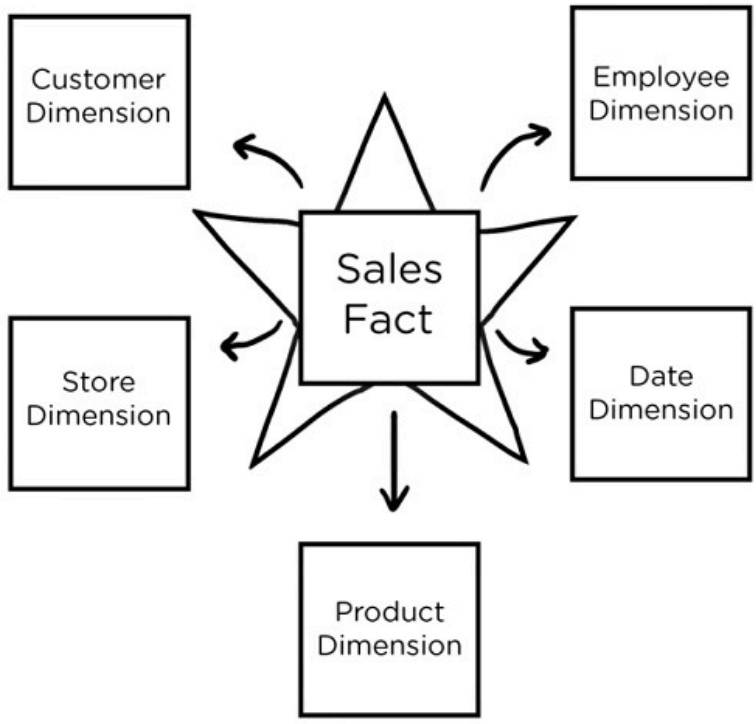
I like an alternative name for constellation schema as multi-star schema, because it has a group of stars.

Please, avoid a spaghetti schema where tables are connected in an unmanageable and complex way, like a spaghetti dish with all the threads tangled together as the opposite of a well-structured star schema data model.

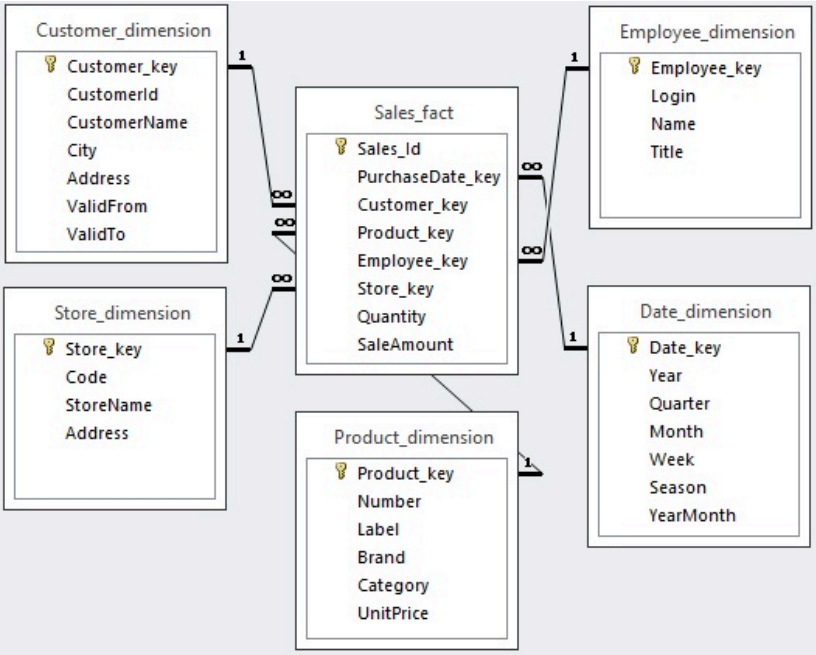
Dimension and fact conformance is a must in a successfull data mart implementation to meet business requirements specification or user story and the requirements of legislation, accepted practices, prescribed rules and regulations, specified standards and terms of a contract as well as best practice.

Sales mart example

Star schema



Entity Relationship Diagram ERD with five one-to-many relationships.



Dimension has a dimension key column with suffix _key as a primary key.
Fact has a sequence number with suffix _Id as a primary key and five dimension keys to each dimension and two measures.

Sales fact has a dimension key PurchaseDate_key as a role-playing date dimension with the role »purchase«.

Sales fact grain is that a single fact table row represents a sale on a purchase date to a customer of a product by an employee in a store. A customer can buy several different products in a sale, which are represented in several rows in the Sales fact table.

Customer dimension is type 2 because it has timeline metadata columns ValidFrom and ValidTo and the other dimensions is type 1.
Read about Kimball dimension type 0, type 1 and type 2 in section 4.3 called slowly changing dimensions.

Date dimension is a type 0 with original values (constant values) and a dimension key in column Date_key of data type date for the primary key of the dimension. Other columns create a hierarchy as Year→Quarter→Month→Date (four levels) and Year→Week→Date (three levels). Other column creates a group for Season.

Date_key	Year	Quarter	Month	Week	Season	YearMonth
2014-04-23	2014	2014Q2	April	201417	Spring	201404
2014-04-24	2014	2014Q2	April	201417	Spring	201404
2014-04-25	2014	2014Q2	April	201417	Spring	201404
2014-05-08	2014	2014Q2	May	201419	Spring	201405
2014-08-16	2014	2014Q3	August	201433	Summer	201408

Employee dimension is a type 1 with most recent, current values and a dimension key in column Employee_key as an artificial auto-generated unique sequence number or an identity column as a surrogate key for the primary key of the dimension and some context descriptive data about an employee.

Employee_key	Login	Name	Title
1	NADA	Nancy Davolio	Sales rep
2	JALE	Janet Leverling	Sales rep
3	MISU	Michael Suyama	Sales manager

Store dimension is a type 1 with most recent, current values and a dimension key in column Store_key as a unique sequence number as a surrogate key and some context descriptive data about a store.

Store_key	Code	StoreName	Address
13	UK-LON	Shophouse	14 Oxford Street, London
16	FR-PAR	Boutique	Rue Cler 52, Paris

Product dimension is a type 1 with most recent, current values and a dimension key in column Product_key as a unique sequence number as a surrogate key. Other

columns create a hierarchy as Category→Brand→Label (three levels).

Product_key	Number	Label	Brand	Category	UnitPrice
504	571145292	Blou jeans	Zara	Clothing	60
538	571145311	Comfie pants	Lee	Clothing	25
893	245600390	Walk air	Adidas	Shoes	89
1408	746153022	Passion no 9	Chanel	Beauty	37

Customer dimension is a type 2 to keep history in multiple rows or records when a customer changes name or city, and a dimension key in column Customer_key as an artificial auto-generated unique sequence number or an identity column as a surrogate key for the primary key of the dimension. Customer_key values are unique for each row even though the same customer is standing in several rows e.g. CustomerId 421 Marie.

Marie Beaulieur as her maiden name was valid from she was born in Lyon or from she becomes a customer and that date is known in a source system, and she is inserted into a first row with ValidFrom as 1900-01-01 for a beginning of time.

Marie Beaulieur got married at 2007-11-16 and took her new husband's surname, therefore her name is changed to Marie Lesauvage and she moved to Paris, and she is inserted into a new row.

Marie Lesauvage remarried at 2014-07-15 and took her new husband's surname, therefore her name is changed to Marie Sainte and she moved to Nice, and she is inserted into a new row.

Timeline metadata columns ValidFrom and ValidTo represents the span of time when a row was the »current truth« in an active period as a time window for a customer in a given state, which are read in a row as follows:

»From and including« the date value in ValidFrom column and
»to and not included« the date value in ValidTo column.

Customer_key	CustomerId	CustomerName	City	ValidFrom	ValidTo
1	176	Hans Andersen	London	1900-01-01	9999-12-31
2	359	Joe Watson	Boston	1900-01-01	9999-12-31
3	421	Marie Beaulieur	Lyon	1900-01-01	2007-11-16
4	421	Marie Lesauvage	Paris	2007-11-16	2014-07-15
5	421	Marie Sainte	Nice	2014-07-15	9999-12-31

Marie's name was Marie Lesauvage from and including 2007-11-16 and to and not included (until) 2014-07-15 because she changed name starting from 2014-07-15.

Column CustomerId is a business key and is the »glue« that holds the multiple rows together for a specific customer even though data change over time.

Customer dimension type 2 has not here a metadata column IsCurrent with two values: 0 for historical and 1 for current to mark each data row, instead column ValidTo with value 9999-12-31 will represent the most recent, current values.
For example, with a metadata column IsCurrent in a type 2 Customer dimension:

Customer_key	Custo-merId	CustomerName	City	ValidFrom	ValidTo	IsCurrent
1	176	Hans Andersen	London	1900-01-01	9999-12-31	1
2	359	Joe Watson	Boston	1900-01-01	9999-12-31	1
3	421	Marie Beaulieur	Lyon	1900-01-01	2007-11-16	0
4	421	Marie Lesauvage	Paris	2007-11-16	2014-07-15	0
5	421	Marie Sainte	Nice	2014-07-15	9999-12-31	1

Since Customer dimension is a type 2 it can answer questions like »How many customers moved last year?« or »How many new customers did we get by month?«.

Sales fact with five dimension key columns for five dimensions where Purchase-Date_key represents the date of purchase, and together with columns for two measures or metrics. A small excerpt from year 2014 where Sales fact contains purchases since 2005 to today. The first two rows tells, that a customer has bought two different products on the same day in the same store and served by two employees.

Purchase-Date_key	Customer_key	Product_key	Employee_key	Store_key	Quantity	Sale Amount
2014-04-23	2	893	2	13	1	89
2014-04-23	2	1408	1	13	1	37
2014-04-24	4	1408	3	16	2	74
2014-04-25	4	504	3	16	1	60
2014-05-08	1	893	1	13	3	267
2014-08-16	5	1408	3	16	1	37

Since I keep history of customers name and cities in the Customer dimension as a type 2, the Sales fact tells us the name of a customer and where a customer lived **at the date of purchase**, when I join a fact row column Customer_key value to the Customer dimension key column Customer_key value for further information about the customer.

For example, at Sales fact column PurchaseDate_key row value 2014-04-25 has Customer_key value 4, which I lookup in Customer dimension to see it is the customer Marie that at that time had surname Lesauvage and was living in Paris when she purchased the product Blou jeans from the employee Michael Suyama in the store Boutique at Rue Cler 52, Paris.

For example, at Sales fact column PurchaseDate_key row value 2014-08-16 has Customer_key value 5, which I lookup in Customer dimension to see it is the customer Marie that at that time had surname Sainte and was living in Nice when she purchased the product Passion no 9 from the employee Michael Suyama in the store Boutique at Rue Cler 52, Paris.

Fact grain
Sales fact grain is that a single fact table row represents a sale on a purchase date to a customer of a product by an employee in a store.

For example in a sql statement to fetch rows that is a grain breakage:

```
SELECT PurchaseDate_key, Customer_key, Product_key, Employee_key, Store_key,
    NumberOfRows = COUNT(*)
FROM Sales_fact
GROUP BY PurchaseDate_key, Customer_key, Product_key, Employee_key, Store_key
HAVING COUNT(*) >= 2
ORDER BY 6 DESC
```

An example of two rows in the sales fact that give a grain breakage because a customer made a purchase of the same product in the morning and in the afternoon from the same employee in the same store, which goes against the grain of the fact table.

Purchase-Date_key	Customer_key	Product_key	Employee_key	Store_key	Quantity	Sale Amount
2014-09-11	5	504	3	16	1	60
2014-09-11	5	504	3	16	2	120

To comply with grain, the two fact rows must be added together in sales fact table.

Purchase-Date_key	Customer_key	Product_key	Employee_key	Store_key	Quantity	Sale Amount
2014-09-11	5	504	3	16	3	180

Sale or Sales
Sale is the selling of goods or services, or a discount on the price.
Sales is a term used to describe the activities that lead to the selling of goods or services. Goods are tangible products as items you buy, goods item, such as food, clothing, toys, furniture, and toothpaste. Services are intangible as actions such as haircuts, medical check-ups, mail delivery, car repair, and teaching. A sales organizations can be broken up into different sales teams based on regions or type of product.

Naming rules for dimension table and fact table and for views upon them
There are several naming rules for a table, e.g. a dimension table name is singular and a fact table name is plural to do a differentiate between dimension and fact. I prefer singular naming for all objects in a database.

A naming rule is to add a role to a table through a suffix in a table name, like Customer_dimension and Sales_fact from the above star schema and Entity Relationship Diagram.

Another naming rule is to add a role to a table through a prefix in a table name, like DimCustomer or Dim_Customer and FactSales or Fact_Sales.

Another naming rule is to add a role to a table through a database schemas, e.g. **dim** for dimension tables and **fact** for fact tables, like dim.Customer and fact.Sales.

A naming rule for views upon dimensions and facts is to add a role to a view through a database schemas, e.g. **dims** for dimension views and **facts** for fact views, like dims.Customer and facts.Sales.

Business users have granted read access permission to the views via database schemas, and no reading access to the tables.

Alternative database schemas like AnalysisDim.Customer and AnalysisFact.Sales.

Extra view naming rule is to add two roles to a view through prefix and suffix, like DimCustomer_Current or Dim_Customer_Current where the prefix tells it is a dimension and the suffix tells the view will present current values from a Customer dimension. With a database schema **dims**, like dims.Customer_Current.

An alternative for dimension views is to place them in four database schemas, like dimcurrent, dimhistorical, dimoriginal and dimregistered and reuse name of view Customer in all schemas plus a Customer_Unique in dimcurrent schema.
Writing a sql statement Inner Join part will use IntelliSense e.g. dimcurrent. and get a list of all dimensions that presents current values in join with a fact table.

Dimension values through views
I call it **registered values** at the date of purchase, and the data is the factual or fact back then, because the Customer dimension is a type 2 that preserves data as it was on a transaction date for a sales process back in time. (In danish »registreret værdi«, a similar term is to maintain posted values »at fastholde bogførte værdier, opslået værdi«).

I create a SQL view upon a type 2 dimension for a **registered view** that present the values from the dimension, view DimCustomer_Registered presents:

Customer_key	CustomerId	CustomerName	City
1	176	Hans Andersen	London
2	359	Joe Watson	Boston
3	421	Marie Beaulieur	Lyon
4	421	Marie Lesauvage	Paris
5	421	Marie Sainte	Nice

View DimCustomer_Registered has dimension key column Customer_key to join on to the Sales fact table or a fact view, and all the values of the dimension key column Customer_key in Sales fact exist in the view.

When I want to make a statistic of where the customers lived at date of purchase, I will use the registered values, e.g. column City from the registered view.

I call it **current values** for all dates of purchases, because the Customer dimension is a type 2 where rows with ValidTo value 9999-12-31 will represent the most recent, current values at present time for a type 2 dimension. (In danish aktuelle, nuværende, gældende, seneste værdi).

I create a SQL view upon a type 2 dimension for a **current view** that present the values from the dimension, view DimCustomer_Current presents:

Customer_key	CustomerId	CustomerName	City
1	176	Hans Andersen	London
2	359	Joe Watson	Boston
3	421	Marie Sainte	Nice
4	421	Marie Sainte	Nice
5	421	Marie Sainte	Nice

View DimCustomer_Current has dimension key column Customer_key to join on to the Sales fact table or a fact view, and all the values of dimension key column Customer_key in Sales fact exist in the view, although the view only shows most recent, current names of the customers. Therefore Marie Sainte is in three rows with different values in Customer_key, because all her purchases with her previous surname (Beaulieur and Lesauvage) now refer to her current surname (Sainte) via the view DimCustomer_Current.

When I want to write a letter to customer Marie, I will use the current values, e.g. columns CustomerName and City (and Address) from the current view which present Marie Sainte and Nice.

For example, at Sales fact column PurchaseDate_key row value 2014-04-25 has Customer_key value 4, which I lookup in the view DimCustomer_Current to see it is the customer Marie with her current surname Sainte and is living in Nice.

Both views DimCustomer_Registered and DimCustomer_Current will present all the values from dimension key column Customer_key, therefore both views can join to the Sales fact table on dimension key column Customer_key without any data loss. Read more about the views in section 6.2.2.

Sales fact dimension key column Customer_key does not tell whether the Customer dimension is a type 1 or a type 2.

When I join Sales fact to view DimCustomer_Registered on dimension key column Customer_key, I tell, that I like to see the customer registered values for a »monthly sales in cities« report over the last twenty years.

When I join Sales fact to view DimCustomer_Current on dimension key column Customer_key, I tell, that I like to see the customer current values for a »marketing planning in cities« report.

It is in the join part from a fact table to a dimension table, I choose which dimension values I want to fetch and see through a _Registered view or a _Current view.

Dimension views
I recommend to have a SQL view for each dimension and two views for type 2 dim. Either with a prefix Dim in view names:

- DimDate or a role-playing dimension DimPurchaseDate, more in section 4.4.
- DimEmployee_Current
- DimProduct_Current
- DimStore_Current
- DimCustomer_Current
- DimCustomer_Registered

Or better with a database schema dims which the business users have granted read access permission to the views to be used in BI tools:

- dims.PurchaseDate
- dims.Employee_Current
- dims.Product_Current
- dims.Store_Current
- dims.Customer_Current
- dims.Customer_Registered

I recommend to have a SQL view for each fact table, e.g. FactSales or facts.Sales, thereby a business user will never access a table in a database, only access data through views in a database.

Example of a statistic of where the customers lived at date of purchase
A »monthly sales in cities« report will use the city of the customers at the time a purchase was happening, occurred, registered or took place, for example:

Month	City	TotalSaleAmount
April 2014	Boston	126
April 2014	Paris	134
May 2014	London	267
August 2014	Nice	37

Example of a statistic of where the customers currently live
A »marketing planning in cities« report will use the current city of the customers, for example:

City	TotalSaleAmount
Boston	126
London	267
Nice	171

Example of implemented views on the Customer dimension and join to fact sales

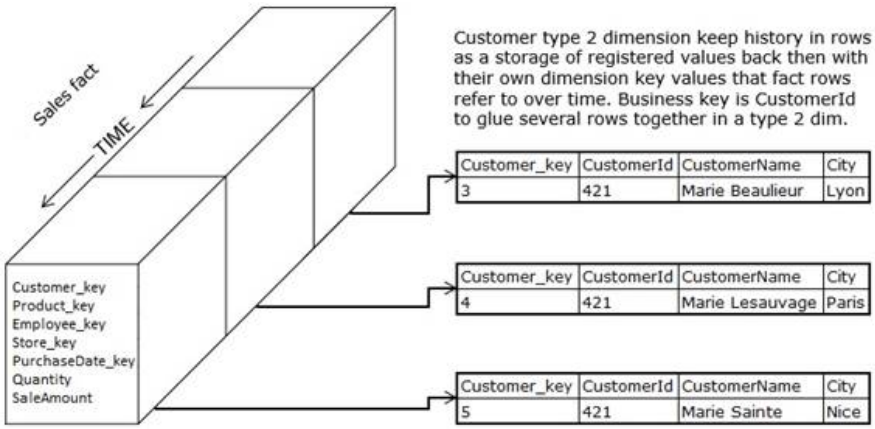
Felt:	Month: MonthYearText	City	TotalSaleAmount: SaleAmount	YearMonth
Tabel:	DimPurchaseDate	DimCustomer_Registered	FactSales	DimPurchaseDate
Total:	Gruppér efter	Gruppér efter	Sum	Gruppér efter
Sorter:				Stigende
Vis:	☑	☑	☑	☐
Kriterier:				

Felt:	Month: MonthYearText	City	TotalSaleAmount: SaleAmount	YearMonth
Tabel:	DimPurchaseDate	DimCustomer_Current	FactSales	DimPurchaseDate
Total:	Gruppér efter	Gruppér efter	Sum	Gruppér efter
Sorter:				Stigende
Vis:	☑	☑	☑	☐
Kriterier:				

```
CREATE VIEW DimCustomer_Current AS
SELECT d.Customer_key, d.CustomerId, c.CustomerName, c.City, c.[Address]
FROM Customer_dimension d
    INNER JOIN -- self join for a type 2 dimension to fetch current values.
      (SELECT CustomerId, CustomerName, City, [Address] -- the most recent data.
       FROM Customer_dimension
       WHERE ValidTo = '9999-12-31'
      ) c ON c.CustomerId = d.CustomerId

CREATE VIEW DimCustomer_Registered AS
SELECT d.Customer_key, d.CustomerId, d.CustomerName, d.City, d.[Address]
FROM Customer_dimension d

-- Monthly sales in cities where customers lived at the time a purchase took place.
SELECT dp.MonthYearText AS Month, dc.City, Sum(fs.SaleAmount) AS TotalSaleAmount
FROM FactSales fs
    INNER JOIN DimPurchaseDate dp ON dp.PurchaseDate_key = fs.PurchaseDate_key
    INNER JOIN DimCustomer_Registered dc ON dc.Customer_key = fs.Customer_key
GROUP BY dp.MonthYearText, dc.City, dp.YearMonth
ORDER BY dp.YearMonth

-- Monthly sales in current cities where customers currently live.
SELECT dp.MonthYearText AS Month, dc.City, Sum(fs.SaleAmount) AS TotalSaleAmount
FROM FactSales fs
    INNER JOIN DimPurchaseDate dp ON dp.PurchaseDate_key = fs.PurchaseDate_key
    INNER JOIN DimCustomer_Current dc ON dc.Customer_key = fs.Customer_key
GROUP BY dp.MonthYearText, dc.City, dp.YearMonth
ORDER BY dp.YearMonth
```

A slide presentation of the above star schema example with data values

Illustration of Sales fact table rows refer to Customer type 2 dimension table rows



Kimball's type 2 dimension fulfills Inmon's data warehouse definition for non-volatile because context descriptive data values do not change for a customer, whereby a fact row refers to the registered values at the date of purchase. When a customer value is »slowly changing«, e.g. changing of name or city values, the Customer dimension will get a new row with the new values, and there will be **no revisiting** of fact rows and no »changing fact«.

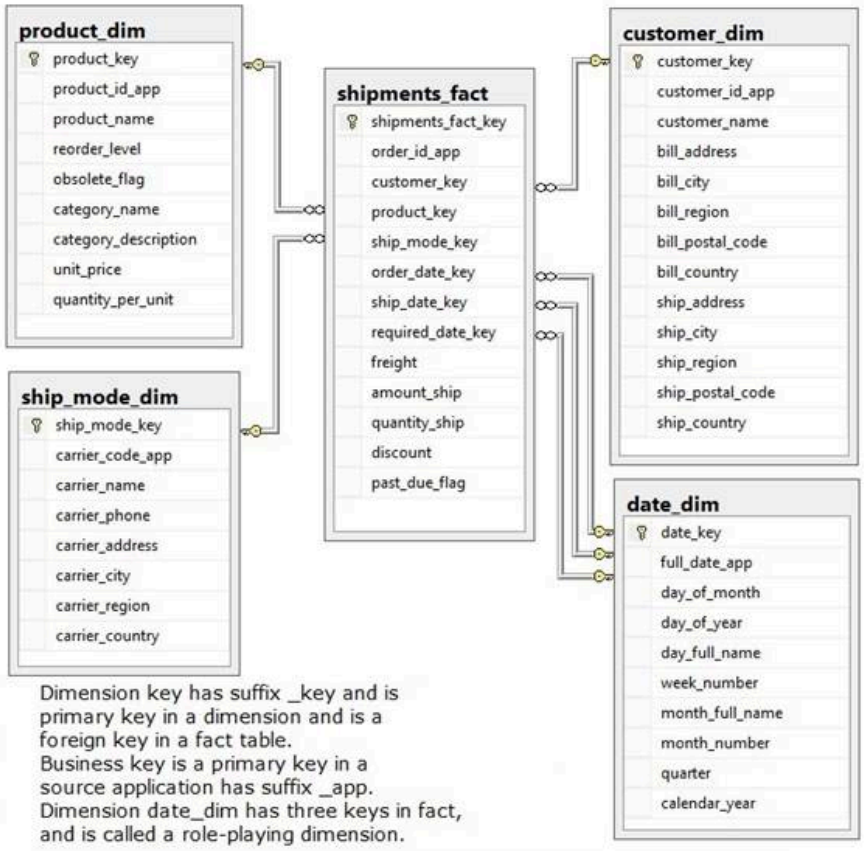
Read about Kimball type 2 dimension in section 4.3 called slowly changing dimensions.
Read examples of Kimball dimension type 1, type 2 and type 7 with most recent, current values in section 6.2.

Improvement expansion
Calculation of SaleAmount in Sales fact is using UnitPrice from Product dimension, and in case of an offer price, it could be a good idea to include UnitPrice, DiscountAmount and OfferPrice in the Sales fact, because Product dimension as a type 1 will overwrite changes of the prices and amounts and only show the current unit price of a product.

We assume that the sales business process does promotions as well, and we have a Promotion dimension with values like »Black Friday« and »New year's sale« and we include it into a Sales fact to cover business needs of information about the sales that happened, or in other words for transactions that happened. However, there are times when a promotion is on but no transaction happens! This is a valuable analytical report for a decision maker, because he will understand the situation and investigate to find out what went wrong with a promotion that doesn't cause sales. We will need a Promotion fact which is very informative because it tells us on which dates there was a promotion at specific stores for specific products. There is no measure related to it, therefore we labelled this type of fact for Factless fact.

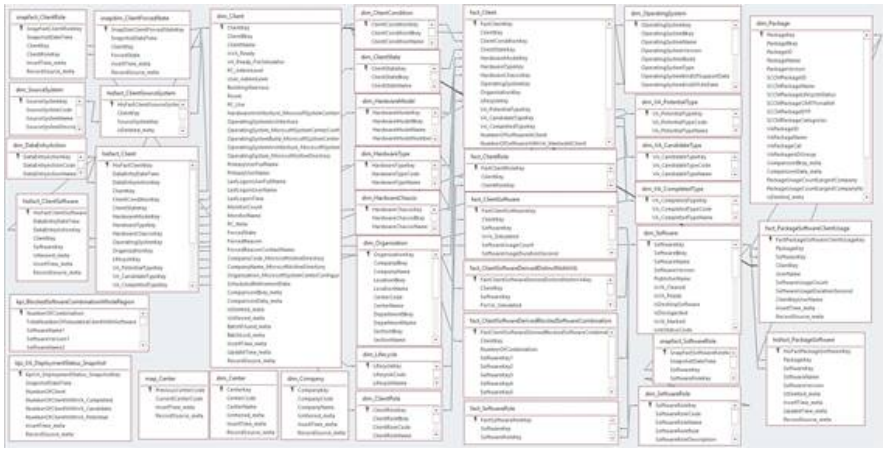
Shipment mart example

Star Schema from Polaris data mart in Microsoft SQL Server database



VAP mart example
VAP stands for Virtual Workplace Portal in danish Virtuel Arbejdsplads Portal.

Constellation schema multiple facts & multiple conformed and shared dimensions



From a real implementation by Joakim Dalby in 2021 based on 65000 devices/end-points/clients and 34000 software (arp string for add remove program) at hospitals in Copenhagen area of Denmark based on four source systems: AD, SCCM/MECM, ServiceNow and Software Central. Dimensions are type 1 with current values where old data is overwritten, and deleted data in a source system will be marked with 1/True value in the dimension metadata column IsDeleted_meta for a soft delete. Facts are based on »capture a relationship in the fact« e.g. fact_Client, and based on derived facts and snapshot facts, read more in section 3.3.

1.10.5. Agile modeling
Inmon modeling is based on a normalized data model and the new data models for EDW wants to break things out into »parts« for agility, extensibility, flexibility, generally, scalability and productivity to facilitate the capture of things that are either interpreted in different ways or changing independently of each other.

Agile modeling has only extension of new tables and none modification of existing tables. Changes in an agile data warehouse environment only require extensions, not modifications and no impact to the data warehouse and it becomes quick-moving to act quickly to easily customize the data warehouse when business changes. An ensemble modeling pattern, e.g. Ensemble Logical Modeling (ELM) gives the ability to build incrementally and future changes should not impact the existing design. Read more.

Agile development is performed in an iterative, incremental and collaborative manner. I think we will always meet an issue that address us to do a model refactoring or re-engineering. Read more.

Inmon modeling is a free data model to do split of keys to mapping tables and data into super-sub tables, to handle one-to-one, one-to-many and many-to-many relationships by connection tables. With Entity Relationship modeling you can make your own labelling of tables in a data model, compared to fixed labels in Anchor modeling and Data vault modeling and Dimensional modeling.

Anchor modeling and Data vault modeling and Dimensional modeling basic idea is to do a **decomposition** of a business entity or several business entities, and to split business keys into their own entities (anchor, hub, key mapping) and to split of all other data to make a new **composition** of entities (attributes, satellites, pits, dimensions, facts and seldome bridges) to form a data warehouse to preserve when data was happening, occurred, registered or took place in operational systems to form a data warehouse according to the definition.

In Anchor model an anchor has multiple attributes attached, and in Data vault model a hub has multiple satellites attached because:

- Multiple source systems divide data in a natural way.
- A large number of columns are separated into smaller subject-oriented attributes or satellites (maybe based on super-sub entity).
- For additional columns due to new business needs in order not to extend existing attributes, satellites and ETL process.
- Different change frequencies of columns and to be 100% insert only compliant.

This has advantages of independent load jobs and data model enhancements but makes it more difficult to fetch and integrate data, e.g. to a Business data area called Business Vault or to dimensions and facts.
[To reduce the complexity in queries with joins, a proven approach is to create (or generate) a view layer on top of the model with Join Elimination.]

A key mapping table has often only one dimension table attached, because a dimension does fetch and integrate data as a result of a merge of multiple source system data and become a conformed dimension and a shared dimension. A dimension has data values that are directly ready to be used by the business users.

It is all about decomposition and composition of data in a data model. Sometimes you don't need to cross the river to get water.

Normalization to sixth normal form 6NF is intended to decompose table columns to irreducible components, a hypernormalized database where essentially every column has its own table with a key-value pair. I see 6NF as a physical level, while I see 1NF to 5NF, you can include DKNF as a logical level useful for design and modeling of a database.

A generic data model does not need any changes when there is a change in the source systems. There is a Party modeling, a 2G modeling and a Focal Point modeling or Focal Framework (YT video and YT serie) and more than dozens of data warehouse data modeling methodologies, approaches, techniques and design patterns that have been introduced over the past decade.

EDM = Enterprise Data Model is a term from IBM back in 1998 for a data modeling method for data warehousing, where ETL was called Capture, Transform, Apply or Extract, Clean, Conform, Deliver processes.

Data ingestion or ingesting or ingest is more general compared to ETL and it may or may not include transformations because it is the process of moving data from one or more source locations to a target location optimized for in-depth analysis. Read more.

A simple Guidedance for EDW: Do not lose data (auditing/compliance/validation). Model it, so you can understand it. Make it physical, so you can query it.

Data warehouse is using a surrogate key instead of a business key to remove dependence from a source system, read more in section 4.2.

1.11. Enterprise data warehouse architecture and ETL process

An illustration below of four data warehouse architectures with data flow in a ETL process to handle data in a multi-layer architecture or in a multi-tier database architecture with different **data areas** (data layers or data zones).

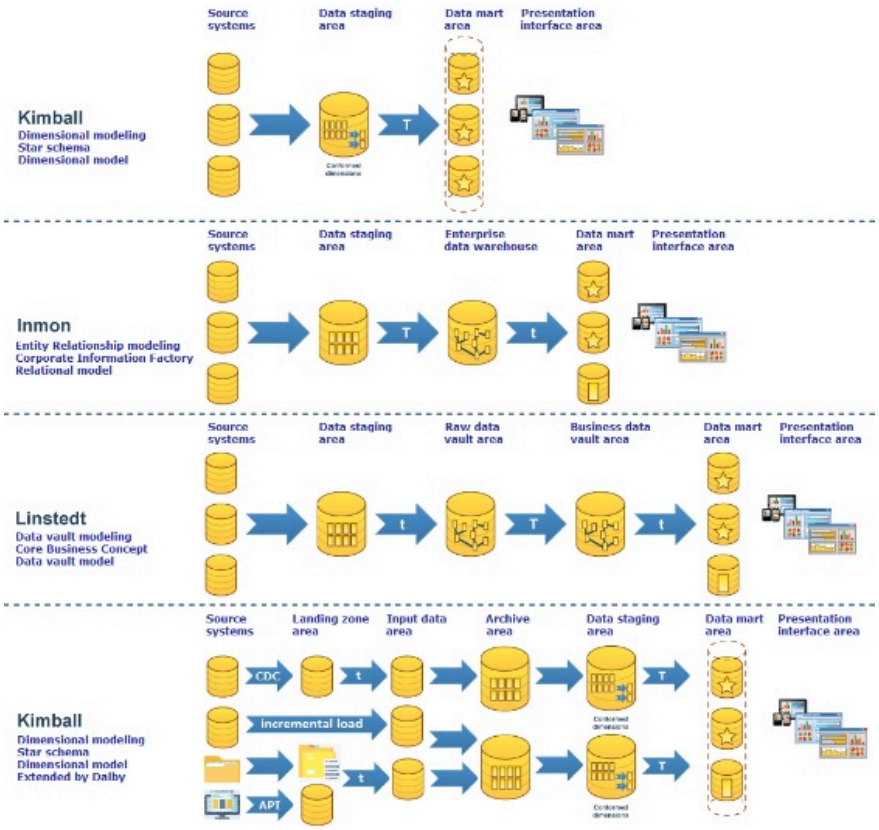
Remarks to the illustration below:

- Kimball stores data in star schema data marts based on the dimensional model. Kimball said: »A data warehouse is a union of all its data marts« illustrated by the dashed line as a Data mart area of several databases with their own specific purpose and tailored data that are business oriented.
- Inmon stores data in an Enterprise data warehouse based on the relational model (3NF) and it provides data to star schema data marts or to other data models in data marts.
- Linstedt stores data in a Raw data vault area (an archive of structure) and a Business data vault area based on the data vault model and it provides data to star schema data marts or to other data models in data marts.
- Kimball extended by Dalby with data areas from section 1.7 with different source data and different ETL processes placed at one data warehouse server, or better placed at several data area servers. Source data is stored in an Archive area (or in a Persistent staging area) through full load, incremental load or delta load, before source data is extracted to a Data Staging area to performing dimensional modeling.

Each data warehouse architecture ends up in several data marts with a dimensional model to feed a dimensionally-structured enterprise presentation area which will consist a business process-centric fact integrated via a set of conformed and shared dimensions in a star schema. Many BI tools in Presentation interface area prefer a dimensional model, where dimensions are used for dropdown box search criteria for a report and for a dashboard or as report front column or column headings.
Data mart can have other data models for specific purposes.
Sometimes called a hybrid architecture.

Dimension and fact tables in a data mart database will focus on ease of business users accessibility where data is easy to understand, users can more readily create the queries they require, and the queries themselves are less complex. In addition, the query response from the underlying dimensional structure will be quicker.

Letter **t** represents a little transform and letter **T** represents more transformation such as integration, preparation, cleansing, uniformity, conforming, enrichment and calculation of source data from multiple source systems.



Inmon's Enterprise data warehouse (EDW) is a common data area before a Data mart area (DMA), and I like to have an extra data staging area between EDW and DMA (not illustrated) for a ETL process and it can be placed inside a DMA database as a local **Data mart staging area** (DMSA). Sometimes a data mart is implemented as sql views which refer to a EDW database as a virtual data mart.
Illustration of a DMSA

Extended by Dalby has been inspired by Data Mesh for a smaller-scale efforts rather than large and enterprise-intensive effort like Corporate Information Factory (CIF) in one Enterprise data warehouse database. It can be called a Federated enterprise data warehouse (FEDW) comprised of multiple independent data marts at one dwh server or at two or several data area servers.

Relational online analytical processing (ROLAP) when a BI tool does direct query through SQL statements to a data warehouse or a data mart to fetch data and display data in a report or a dashboard. It is seldome a performance friendly way.

Therefore between a data mart and a BI tool there is a OLAP cube to handle OnLine Analytical Processing in different models:

- **Multidimensional model**
  also known as a Multidimensional online analytical processing (MOLAP)
- **Tabular model**

For me, it is a principle for a cube that all dimensions and all measures can be combined freely else divide data into multiple cubes.

SQL (Structured Query Language) is a query language for database.
MDX (Multidimensional Expressions) is a query language for multidimensional cube.
DAX (Data Analysis Expressions) is a query language for tabular cube.

Read more about differences of opinion.

ETL process stands for **E**xtract, **T**ransform, **L**oad, see a model, and ETL exists between the data areas (data layers) of a data warehouse solution, e.g.:

Between each data area there is a ETL process to:

- Extract data from the previous data area with criteria to fetch and limit data.
- Transform data by integration, cleansing, conforming, enrichment, etc.
- Load data to the next data area.

It is a common way of handling data in a multi-layer architecture.

(In danish lagdelt arkitektur; extract for tilvejebringe data ved at udtrække, uddrage eller udlæse data; transform for at transformere, bearbejde, behandle, beregne, berige data, sammenlægning, sammenstille og samkøring; load for at levere data eller indlæse data til det næste område).

1.12. Data mart category

A Data mart area (DMA) will consist of several data marts database. Data is fetched from a Data staging area (DSA) or an Enterprise data warehouse (EDW) through a ETL process which may contain in a local Data mart staging area (DMSA), see more in section 1.11.

Each data mart database has a specific purpose for tailored support and sometimes it can be characterized as one of the categories below or types of data marts.

Common mart
With loaded tables of data from DSA or EDW with data across business processes and multiple source systems.
With common dimensions to be reused in the other data marts. Examples of dimensions are Date, Time, Employee, Organization, Product, Retailer and TransactionType. Can also include role-playing dimension, e.g. Manager based upon Employee and different date dimensions with unique names of columns.
With common facts to be reused in the other data marts. Examples of facts are Sales, Inventory, Marketing, Finance, Employment and Transaction.

Consolidated mart

With loaded tables of data from DSA or EDW with data across business processes and multiple source systems. EDW can be many star schemas in first-level data marts with dimensional building blocks with shared, common and conformed dimensions in a constellation schema which is great to have before tackling the task of consolidating to a consolidated mart with consolidated dimensions and consolidated facts in a second-level data mart. See an illustration.
A consolidated data mart is an integrated superset of single business process-oriented dimensional models. It brings together the necessary data from multiple business processes into a single consolidated mart to support a suite of analytical applications. For example, a customer profitability consolidated mart might bring together data from orders, invoicing, solicitations and customer service in order to support customer attrition, customer promotional effectiveness where all the data has been consolidated into a single fact table do to cross-sell analytic applications.
Read more in Kimball page 126 and in Kimball Design Tip #39.

Subject mart
With loaded tables of data from DSA or EDW with data across business processes and multiple source systems. Data can be a subset from a common mart and from a consolidated mart, and data can be further enriched with specific subject data.
For example, Sales mart, Customer mart, CRM mart, Churn prediction mart, Market mart, Procurement mart, Inventory mart, Production mart, Shipment mart, HR mart, Tax mart, Credit risk mart, Fraud detection mart, Financial mart, Budget mart, ESG Environmental, Social and Governance mart.

Analytical mart
With views upon common mart, consolidated mart, subject mart or EDW to fetch relevant data. Making conformed columns to provide data to OLAP cube for Multi-dimensional model or Tabular model to dashboard visualization tools like Excel, Power BI, Tableau, QlikView, Qlik Sense, Alteryx or Targit. The data mart can be used for analytics by business users and data scientist to do ad hoc query in a tool like SQL Server Management Studio, DBeaver, Access, Python or RStudio.

Analysis data area mart
Sometimes called Advanced data analysis or Automated data analysis (ADA).
Sometimes business users want sql query access to dimensions and facts but they don't like to join them together, therefore I implement a either a sql view that refers to a data mart or to join a fact and its associated dimensions together including relevante columns, or a materialized view to store data into a table as an Enriched Fact Table, One Big Table (OBT), Analytical Base Table or Analytical Business Table (ABT). For dimension data in a join with a fact we need to choose between current values and registered values to be inserted into the table.

Reporting mart
With views upon common mart, consolidated mart, subject mart or EDW to fetch relevant data. Making conformed columns to provide data to reports on papir, in pdf files and other display formats like Excel files.
Stored procedures with criteria parameters inputs for example: FromDate, ToDate, Customer, Branche, Product, SalesDevice, TransactionType will fetch the relevant data from the views and present data to a reporting tool used by business users like Reporting Services (SSRS) or Microsoft Access with pass-through query to call a parameterized stored procedure.
With tables for Regulatory requirement return report (in danish indberetning) for financial reporting, because it is desirable to be able to recreate an old report both as it actually looked at the time of creation and as it should have looked given corrections made to the data after its creation. All tables has a VersionId to be used for sql join of tables to ensure all data rows belong to the same accounting.

Operational mart
With loaded tables of data from DSA or EDW with data across business processes and multiple source systems. It is an interfacing layer for other applications not used for reporting purposes. These applications are not interested in the history of data, so I typically built views on top of the archive area or EDW or data mart area.

Delivery mart
With views upon common mart, consolidated mart, subject mart or EDW to fetch relevant data to provide data and to create data in a file format that is customized to the individual data recipient as another IT system internally or externally.

Discovery mart or Exploration mart or Self service mart
With tables of raw data from source systems, or with views upon an archive area or views upon EDW or views upon data mart area to some business users to do data discovery by sql statements. It is a sandbox where users also has write access and can create their own tables for staging data, mapping data, search criteria data, enrichment, calculation, derived data, and the final result data set to be exported to a Excel sheet or to be connected to Power BI, Python or RStudio. The purpose is to prepare a business requirements specification or a user story for extensions to a data warehouse solution. More about analytic sandbox in Kimball Design Tip #174.

Sometimes a data mart is implemented as views on top of a EDW database as a virtual data mart.

Remember to set a process for user management, permissions and authentication to manage and monitor and reduce the risk of cross contamination of data to ensure that a business user only has access to view relevant data for analyzes and reports.

1.13. Data reconciliation

One part of a data warehousing project is to provide compliance, accountability, and auditability. After data capture, please remember to implement a **Reconciliation Summary Report** with the results from your recon e.g. Uncategorized assets, Invalid country codes, Derivatives transactions with missing currency code.

Audit trail (in danish kontrolspor) becomes important for the credibility of a data warehouse. An example from a source system that has a value 0.003589 and export it to a txt file where the value becomes 3.589E-3 in the scientific notation and by a mistake in a ETL process the data warehouse saved and showed the value as 3.589. A contract number 700002848572 becomes 7.00003E+11 and the last part of the value got lost. When reconciliation is built-in a data model and built-in a ETL process, this mistake would be reported and a programmer can fix the import and update his data profiling documentation. It is to improve data quality and audit trail/control track for reconciliation in a data warehouse.

A classic reconciliation is to weigh the truck before it leaves and weigh the truck when it arrives at the destination to make sure that no load has been lost on the ride.

Do reconciling between source systems and data warehouse with reconciliation of row count and sum of values and mark as reconciled and do auditing report.
(To reconcile in danish at afstemme, stemmer overens, afstemning, kontroloptælling).

Log of RowCount of target rows before a process and after a process to measure the changes in number of rows the process did (TableRowCountBeginAt and TableRowCountEndAt). RowCount of source data rows and extracted rows based on join and criteria. RowCount of target deleted rows, updated rows and inserted rows/loaded rows successfully or rejected rows with missing value or out of bounds amount or other kind of invalid data depends of the validation check of data quality.

Do auditing report to monitor and control to avoid discrepancy between source systems and the data warehouse and make sure that all data have been fetched and saved in the areas of the data warehouse by doing validation check.

Do alert the data owner and data users by email.
Auditing is a kind of inspection (in danish eftersyn).

It is important to do a log of **number of rows and sum of values** from a source systems to the Landing zone area and to Input data area for later **reconciliation check count auditing** in case a source system does not deliver the expected number of rows and to monitor over time the number of rows to see if it increases as expected and be used in a graph over the amount of data in the data warehouse. Other data values e.g. sale amount, quantity and volume can also be logged and monitored.

I recommend that a source system tells the number of rows per table that will be compared to the saved number of rows in Input data area in an audit trail. It can be implemented as a view in a source system.
For example in a sql statement:

```
CREATE VIEW BI_Audit_NumberOfRows AS
SELECT 'Northwind' AS Sourcename, 'Customers' AS Tablename, COUNT(*) AS NumberOfRows
FROM dbo.Customers
UNION ALL
SELECT 'Northwind' AS Sourcename, 'Orders' AS Tablename, COUNT(*) AS NumberOfRows
FROM dbo.Orders
UNION ALL
SELECT 'Northwind' AS Sourcename, 'Order Details' AS Tablename, COUNT(*) AS NumberOfRows
FROM dbo.OrderDetails
```

By sum of values you can reverse the sign in every other row by multiplying each value with -1 to avoid the sum becoming larger than the data type.
For example in a sql statement:

```
;WITH InvoiceItemSummarized AS
(
 SELECT InvoiceItemAmount, Sign = IIF(ROW_NUMBER() OVER(ORDER BY (SELECT 1)) % 2 = 0, 1, -1)
 FROM InvoiceItem
 ORDER BY InvoiceItemId
)
SELECT SUM(InvoiceItemAmount*Sign) AS InvoiceItemAmountSummarized
```

```
FROM InvoiceItemSummarized
ORDER BY InvoiceItemId
```

Reconciliation between data areas or data layers or data zones from source systems to presentation area is important to know if all data are included or some data are disappeared, maybe because of wrong json delivery or a wrong incremental load and delta load between the areas or removing duplicates.

How else can we say, we are compliant and ensure compliance without data governance, data management and data reconciliation, data quality and data lineage mixed with documentation in a bank to meet Basel I, Basel II, Basel III, Basel IV, EDA, HIPAA, Sarbanes-Oxley and BCBS239 (The Basel Committee on Banking Supervision 239) together with a **regulatory requirements return report**.
BCBS239 paper and Explanation and elaboration of Data Governance for BCBS239 to be complient for Danish Financial Supervisory Authority (Danish FSA) (in danish Finanstilsynet).

1.14. Data quality

Data quality purpose is to ensure that business users trust and have confidence to data to achieve reliability and relevance. Defining and implementing processes to measure, monitor and report on data quality, measurement results of data quality performed on a data set and thus for judging a data set, and hopefully to improve the quality of data. The specific characteristics or dimensions of data that are analyzed in a data quality program is differ from one business to another based on the needs and priorities of that business. We must be able to rely on the accuracy of our data on which we base decisions. (In danish »Vi skal kunne stole på korrektheden af vores data, som vi baserer beslutninger på.«) Doing data quality increases believability, credibility and reliability (in danish troværdighed, pålidelighed).

Data quality skills and data cleansing for a better **data discipline** and to avoid violates of rules of data quality with the following dimensions are commonly used as data controls with the **goal** of ensuring data quality, and where one **mean** is conformance or conformity to make data from multiple source systems conformed through data conformation as a part of a ETL process. The term data quality dimension refer to measurement of physical objects, e.g. weight, height, length, width we do call dimensions, what should be measured and reported on for a data set. A data quality dimension defines a set of attributes and characteristics that represent an aspect of the data quality.

The following **twelve data quality dimensions** are significant in relation to assessing the quality of data and to declare a data set with dimensions for data quality. They can be used to cover the need to be able to document that the content of data in a data warehouse is correct:

Timeliness and Availability (aktualitet og tilgængelighed)
Where data is available and accessible when needed by the business. Data is up-to-date. Timeliness is a measure of time between when data is expected versus made available.

For example, a business must report its quarterly results by a certain date, the delay between a change of a real world state and the resulting modification of the data warehouse, and a customer contact information is verified at least once a year and it is indicated with a verification date that data is up-to-date. Another data quality rule can be that max 80% of the active companies must have annual accounts that are not older than 15 months old.

Timeliness depends on the update frequency of a data set in relation to how often the data changes. A data set is updated real-time, daily, weekly, monthly, quarterly, semi-annually, yearly. Currency measures how quickly data reflects the real-world concept that it represents.

Timeliness is also about how frequently data is likely to change and for what reasons. Mapping and code data is remain current for a long period. Master data needs to be up-to-date in real-time. Volatile data remains current for a short period e.g. latest transaction or number of items in stock can be shown with a as-of time to tell business users that there is a risk that data has changed since it was saved/recorded. Typical during a day data will be changed several times and in evening and night data will remain unchanged. Latency tells the time between when data was created or changed and when it was made available for use by the business users. Kimball talks about low latency data delivery can be very valuable for a full spectrum of data quality checks (Kimball page 261, in danish reaktionstid, too fast fact data delivery can cause inferred members, read more in section 4.5).

Completeness (komplethed, fuldstændighed)
Where a data row has the necessary values present for meaningful conclusion.

For example, an address must include house number, street name, zip code, city, country.

Where a data set has the required values to avoid missing values, e.g. an order must include: order number, ordered date, customer number to an existing customer, product, quantity, unit price, order amount, etc.

A data quality rule can check if some columns are empty (null, empty string, 0 not expected). A ETL process must be robust and must not cause system crash and system error at an unexpected null, instead the data row must be sent to a Wrong table which the operation monitors and reports on. When adding new data rows, update or delete rows, it can be checked whether the number of rows corresponds to the expected number of rows e.g. between 100 and 130 new orders per day. The number of new daily transactions does not decrease by 10% compared to the average number of transactions in the latest 30 days (or does not exceed 10%). Keep an eye on the amount of data, the quantity or volume of available data is appropriate.

RowCount of source data rows and extracted rows based on join and criteria.
RowCount of target deleted rows, updated rows and inserted rows/loaded rows successfully or rejected rows in Wrong tables.
RowCount in fact table of artifact values or inferred members, e.g. -1 for »Missing«, -2 for »Not available«, -3 for »Not applicable« and so on to give a warning when the number of rows are too high.
RowCount of out of bounds amount or other kind of invalid data depends of the validation check of data quality. Read about data reconciliation in section 1.13.

Incomplete data e.g. missing one hundred thousand data rows compared to last payload, an address does not include a zip code or an email address is missing the domain (gmail.com) makes the address not usable. A data quality goal could be 98% usable addresses and must mark the data row to be taking care of afterwards. Over-completeness means that a set of rules is too restrictive and value combinations that are actually valid are unjustly excluded.

Uniqueness (entydighed)
Where data identifies one and only one entry. No duplicate rows within a data set. Unique data means that there is only one instance of a specific value appearing in a data set, so it is free from data duplication by removing duplicates.

For example, a Social Security number (SSN) to ensure that each person has a unique Id, therefore duplicate SSN values are not allowed within the data set.

Unambiguous value for having one meaning and avoid ambiguous values at same level in a dimension or need to combine two values to one unique data. Duplication where customer has addresses spelled in different ways is not credible. What about Bob and Bbo, a mistyping, a typos or a misspelling? What about Daniel and Dan may well be the same person? Investigate that duplicates are unique.

Check for fact grain breakage with a simple sql statement `HAVING COUNT(*) >= 2` to fetch rows that is a grain breakage so they can be deleted from the fact table or the fact table is not displayed to the business users until grain is fine again because it makes it impossible to analyze individual transactions and an aggregation or a summarization will provide wrong values in a dashboard or report.

Validity (validitet, gyldighed)
Where data is valid with a range of values and is conformance to rules.

For example, an age of a person cannot contains a negative value and cannot be higher than 125 years. When an age is under 18 then marital status must be not married. A validation rule in a time registration system could be that hours worked must be between 0 and 168 per month as a plausibility range. The average price of this month must not differ from last month's price by more than 50%. A valid data value can also include a regular expression patterns as a column of text has to be validated, e.g. phone number pattern: (999) 999–9999 where we want the hyphen to be stored in the column. A validation rule can characterize as: in-column, cross-column, in-row, cross-rows, cross-data set, like a cross-column validation is about certain conditions that span across multiple columns must hold, e.g. a patient's Date of Discharge from the hospital cannot be earlier than the Date of Admission, or a Delivered date of an order cannot be less than its Shipment date.

Validation ensures that data is logical and conform. Types of validation rules:
Equality, inequality, logical rule mixed with range checks, bounds fixed or bounds depending on entries in other columns. A Due date within an acceptable range and not ten years wrong, else we have **invalid data** that should be marked as rejected to be corrected. It is also important that a data row indicates the period in which it is valid.

Conformity (overensstemmelse)

Where data is stored in a column with a business-wide-understanding userfriendly name and where data is in the required format and following standard data definition.

For example, an easy name of a column for an invoice amount that is excluded VAT (pre-VAT) could be InvoiceAmount_ExclVAT and for the amount included VAT could be InvoiceAmount_InclVAT. Formatting telephone numbers with or without country code and a control have to make sure that all phone numbers has proper number of digits and a format to ensure conformance. Format of a date as yyyy/mm/dd, dd.mm.yyyy, dd-mm-yyyy or yyyy-mm-dd, please choose to have only one format in the data warehouse. How to present workload per day as 7½ hours, you like 7:30 or 7.50?

Uniformity (ensartethed)
Where data is stored in a column with a data type and size and where a value has the right unit. For a data warehouse with data across domains from multiple source systems it is important to define and use the same data types and sizes.

For example, a decimal number in Europe is using comma before decimal (3,14) and US is using period (3.14). How many digits in an amount e.g. data type decimal(19, 4) is a popular choice for e-commerce but Bitcoin is using decimal(24, 8). How large can the sum of an amount column over many rows going to be?

A data value is often represented with a specific unit of measure (UOM), but seldom the unit can been seen in the name of a column or in an extra column with the unit as text. unit of measure examples of a weigth value in pounds or in kilograms, a length value in centimeters (cm) or in feet (ft), a distance value in kilometers (km) or miles (mi) or light year (ly), an amount value has a currency like USD, EUR, DKK, and a temperature value was measured in Celsius ºC or Fahrenheit ºF.

Consistency (konsistent)
Where the same data from two or more source systems must not conflict with each other. Value consistency is to store a value in the same way.

For example, a date as yyyy-mm-dd, or a zip code is right and has a cityname, or a data mapping like DK for Denmark and »Cph, Kbh, København« for Copenhagen to be consistent for the capital of Denmark. Code values must follow different standards, e.g. ISO Currency Code USD, EUR, GBP, DKK, or LEI Legal Entity Identifier issuer code, or NACE codes and CIC codes. Improve values that take care of case-sensitive words through translate, mapping or bridging rules to create one and only one truth.

When a product is discontinued there must not be any sales of the product.

Three data sets referring to the same time period, e.g. one includes data for females, one for male and one for total, the consistency between the results of the three data sets can be checked.

Inconsistency data stands for conflicting data or when the same data is not the same across source systems.

For example, an address is the same like »Fifth Avenue«, »5th Ave« or »5 AV«.
The name of a product with two kind of spellings like Ketchup and Catsup.
Sometimes a user type-in some extra data in an address column with a direction, e.g. »on the corner of Fifth and Main«.
Hopefully a data cleansing processing can catch it.
Inconsistent data gives problem in any database and data warehouse.

Temporal Consistency to ensure that data reflects a consistent state over time according to time-variant from the data warehouse definition. For example, a sales system where a business user queries historical sales data for a specific date, it is expected to provide consistent information.

Cross-system consistency (distributed consistency) is to compare source system data and data warehouse data, e.g. the daily number of transactions must match the number of rows in a fact table. I.e. pertains to ensuring data remains consistent across different systems that interact with each other. In distributed environments, ensuring cross-system consistency prevents data anomalies and conflicts that can arise from concurrent updates.

Structural consistency is to compare database schema for names of tables, columns, data types, relationships and constraints, e.g. in a source system a column data type has been changed from integer to decimal without telling it to the data warehouse team. One day in future the source system has value 3.14159 and send it to the data warehouse where a ETL process will give an error (failure) when load the value into a column that expect an integer, and will raise an error message »Data type mismatch« or »The data type of target column does not match the data type of the source column«. Read more.

Integrity (integritet, pålidelighed)
Where constraints is a not null column, a primary key, a foreign key referential integrity where data stating that all its references are valid. Relationships between data in different systems is maintained to be accuracy and consistency so data relationships are valid and connected. Wrong data type for a data value like too big amount or too long string text. Allowed values like dates, weekdays and week-numbers is constant data.

Accuracy (faktuelt, kendsgerning, nøjagtighed, korrekthed, rigtighed)
Where data represents what it is intending to represent in the real world. Data will correctly reflects the real world.

For example, a Date of Birth (DOB) and age of a person is trusted or an address (Mailing Address or Residence Address because they are not the same for some people) is the correct one by performing a sample measurement, or check and compare an address with another data set known to be accurate, e.g. a National Address Database or openaddresses.io to make a degree of which data correctly describes the real world object or event. A new product name may not have the same name or similar sounding to a competitor's product. A new product claims to weigh a specific amount, and the weight is verified in an independent test.

Inaccurate data or incorrect data or wrong data is poison for a data warehouse, therefore it is good to set up some rules to find inaccurate values and mark them to be taking care of.

Semantic accuracy must ensure consistency between the content of the data recorded and actual conditions (content accuracy).

Syntactic correctness must ensure that data complies with syntactic rules, e.g. spelling and formats, a date or an xml structure (form accuracy, syntax and schema validation). A nurse entered the Date of Birth of all the patients in the format of dd/mm/yyyy instead of the required format of mm/dd/yyyy. The data passed the system validation check as the values are all within the legal range. However they are not accurate.

Reasonability, Anomalies and Outliers (rimelighed, uregelmæssighed, afvigelse)
Where data is based on comparison to benchmark data or past instances of a similar data set.

For example, sales from the previous quarter to prove that data is reasonableness or data has anomalies, or outliers (deviations) with data point or data value that are completely off, e.g. a unit price more than $5000 is unrealistic in a pet shop.

It is a part of a data reconciliation of a data set.

Confidentiality and Security (fortrolighed og sikkerhed)
Where data will be maintained according to national and internatinal standards for data, e.g. GDPR. Which access to data is controlled and restricted appropriately to maintain its security and how long it is retained.

Clarity and Usefulness (klarhed, anvendelighed, forståelighed, genbrugelighed, gennemsigtighed, ansvarlighed)
Where to assess the extent to which data is understandable and can be able to read and used by others without any misunderstanding the data and without difficulty. Presentation quality and usability is the data readability and understandable, relevantly, accessible, maintainable and credibility. Availability of documentation and columns of metadata in a data set.

Reports must have accessibility and communicate information in a clear and concise manner. Reports should be easy to understand yet comprehensive enough to facilitate informed decision-making. Reports should include an appropriate balance between data, analysis and interpretation, and qualitative explanations. Reports should include meaningful information tailored to the needs of the recipients.

Monitoring (tilsyn) the 12 dimensions of data quality
Monitoring the quality of the data and reporting it to the data owner is part of been compliant. Being aware of quality of your data can be termed as data quality but sometimes it is misunderstood that this will improve the actual quality of data. The thing it does is to improve the awareness.
Data quality measurements are established on the basis of the importance and use of data, respectively. Data should be monitored to ensure that it continues to meet requirements.

A data quality indicator is called a metric, and several metrics will behind one data quality dimension. It must be possible to measure each data quality dimension with a limit value range for a data quality rule to indicate whether the measurement is:

- Okay (green)

- Attention-demanding (yellow)
- Action-demanding (red)

A data quality measurement should preferably be performed automatically and data quality measurements should be performed after each ETL process. The result of metrics of data quality measurements must be stored and documented so that they can be exhibited to the data owner and data users, especially with which measurements exceed limit values. The measurements must be able to be included in an overall reporting and follow-up on data quality.

Data Flow Monitoring (DFM) - Measuring the health of a data warehouse system Monitoring the source data on the way to a data warehouse to ensure Timeliness and Availability, and to ensure whether the ETL processes are running or have run within the expected process time, etc.

Do alert the data owner and data users by email.
Auditing is a kind of inspection (in danish eftersyn).

Data quality metrics in a reliability analysis report
Some measurable examples.

Accuracy metrics	To measure how accurate the data sets are
Error rate	Percentage of data points that are incorrect
Matching rate	Percentage of data points that match a known source of truth
Mean absolute error	Average difference between data points and their true values
Completeness metrics	To measure the proportion of missing data within a data set
Missing value percentage	Percentage of fields with missing values, inferred member -1
Completion rate	Percentage of rows with all required fields filled, no inferred
Row count ratio	Ratio of complete rows to total rows
Consistency metrics	To measure whether data adheres to predefined rules and formats
Standardization rate	Percentage of data points conforming to a specific format
Timeliness metrics	To measure the freshness and relevance of data
Data age	Average time elapsed since data was captured or updated
Latency	Time taken for data to be available after its generation
Currency rate	Percentage of data points that reflect the latest information
Uniqueness metrics	To ensure all rows are distinct and avoid duplicates
Unique row rate	Percentage of rows with unique identifiers
Duplicate row rate	Percentage of rows that are identical copies of others
Deduplication rate	Percentage of duplicate rows identified and removed
Outliers metrics	To measure whether data adheres to predefined value ranges
Outlier rate	Percentage of data points that deviate significantly from the norm

Remarks
- Correction like changing a wrong year of a date or casted to right data type.
- For fact tables one truth depends of the use because a fact for Finance and a fact for Marketing may look like the same but their business requirements specification has small differences.
- Cross-contamination in a ETL processes by which bad data is unintentionally transferred from a source system to the data warehouse with harmful effect on data quality.
- Data profiling is a classic discipline to ensure a data understanding and to identify the current data quality and maybe to remove data that isn't accurate and/or isn't relevant, e.g. duplicates based on a rule of unique data.
- For a data warehouse tasks as cleansing and scrubbing is ways to improve the data quality in the act of removing wrong data, incorrect data, incomplete data and duplicated data, the unwanted data is discarded.
- Other data quality dimensions could be about precision, availability and credibility. Saying that you live on the earth is accuracy, but not precise. Where on the earth? Saying that you live at a particular address is more precise.
- Quality assurance is a nice side effect to over time to improve other systems.
- Data is fact, not truth, please keep your intuition and common sense and not only obey what the data tells you.
- In danish Digitaliseringsstyrelsens Fælles sprog for datakvalitet er en standard omkring datakvalitet i fællesoffentlige projekter.

Forty-three data quality dimensions Twenty data quality dimensions from 1991

Simplify data quality validation with SQL

There are two types of strategies for improving data quality:
- Data-driven by modifying the data value directly.
- Process-driven by redesigns the process which is produced or modified data in order to improve its quality.

Process-driven is better performing than Data-driven in long period, because it remove root causes of the quality problems completely.

Microsoft Purview for data quality

Vilfredo Parento from Italy made in 1906 the 80:20 rule or principle built on observations of his such as that 80% of the wealth in Italy belonged to about 20% of the population. The same ratios (in danish forholdstal) is often in data quality that 20% of a data set generates 80% of the errors in data. Corresponding to that 20% of the goods generating 80% of the earnings, which means that the other 80% of the goods generate only 20% of the earnings.

1.15. Data lineage and Data provenance

Information about where the data came from and how it was transformed.
Data lineage is the journey data takes from its creation through its transformations over time. Data lineage can tell us the source of data, where did the data come from and what happened to the data and what transformations took place, and where data moves to over time. It describes a certain data set's origin, movement, characteristics and quality, and what happens to it and where it moves over time.

Data lineage gives auditability and traceability in a data flow, and visibility while greatly simplifying the ability to trace errors back to the root cause in a data analytics process. Lineage is about achieving traceability of data to ensure that any given data point can easily be traced back (find a way back) to its origin.
(In danish afstamning, hvor stammer data fra eller hvor kommer data fra samt hvor anvendes data, sporbarhed i data strømmen).

For example, insert, update, load timestamp and a detailed record source are required for every row that arrives in a data warehouse and about the processed data.

Data lineage is a end-to-end mapping of upstream and downstream dependencies of the data from capture or ingestion to analytics including which reports and dashboards rely on which data sources, and what specific transformations and modeling take place at every stage. Data lineage also includes how data is obtained by a transformation. Data lineage must ensure that there is traceability from where data is created and where data is subsequently used. Data lineage is both at table level and column level.

When a data warehouse is using data from other source systems, inhouse or external, a data lineage can be implemented by tagging data rows with audit data in a column **RecordSource** from Data vault or column SystemOfRecord to indicate where the data originated.

When data is staged and present in a target place in a data warehouse, data lineage is also important to inform a business user where data is comming from. If an organization does not know where data comes from and how it has changed as it has moved between systems, then the organization can not prove that the data represents what they claim it represents.

In addition to documentation, data lineage is also used in connection with change management so that a data steward can determine which IT systems and solutions will be affected and how they will be affected.

Data provenance focuses on the origin of the data aka source system, could be a screen from a host system where users type-in data values after a phone conversation with af customer, or customer fills a web page. The original data capture. Data provenance is responsible for providing a list of origin, including inputs, entities, systems, and processes related to specific data. Always good to know the latest date of update and who made the update.

If you want to drive real value from your data, you must first understand where data is coming from, where data has been, how data is processed, how data is being used, and who is using data. That’s what data lineage and data provenance is all about to create transparency.

1.16. Documentation

My experience is that documentation of a data warehouse system takes at least ϖ (Pi) times as long as the development time. Tasks like writing explanations, reasons and arguments, set up tables, drawings, figures and diagramming, show data flow and data lineage and make a data dictionary, and last but not least proofreading. Followed by review by colleagues and subsequent adjustments and additional writings. Documentation is an iterative process and in a good agile way a programmer should program in the morning and document it in the afternoon.
There are many tools e.g. Erwin or Collibra or Sqldbm and Lineage tool to show a change in one system has effects in another system.

1.17. GDPR

For EU GDPR General Data Protection Regulation (Persondataforordning) a business key like a Social Security Number needs to be anonymous after some year by adding a calculated cross sum of the SSN and hash it. It will be a nice scramble or encryption by a scrambling algorithm. To be sure of total anonymity only save Date

of Birth and Gender of a person; no name, no address, no zip code and no city. A Master Data Management (MDM) database can contain social security number, name, address together with CustomerId that is used in emails to customers.

Therefore we have a surrogate key CustomerInternalId that is used in all other operational systems and data warehouse together with BirthDate and Gender which is not personally identifiable or enforceable. A web service will provide all systems the data of personally from MDM database. When a customer is going to be deleted because it is his/her wish or the data limitation period, we only need to anonymization (the way of delete data) in the MDM database and we can keep CustomerInternalId, BirthDate and Gender in the other systems to make sure statistics remain unchanged back in time. When a system is calling the web service it will give back unknown for SSN, name and address when a customer no longer exists in MDM database. If we don't have a MDM, we must do an anonymization in all data areas and in relevant source systems as well, and flag data with an IsAnonymized column. Personal Information (PI), Personally Identifying Information (PII), Sensitive Personal Information (SPI) is data relating to identifying a person. Read more.

1.18. DataOps and Data governance and Data management

DataOps is how to organize your data and make it more trusted and secure with tools like:

- Data governance - Catalog, protect and govern all data types, trace data lineage, and manage data lakes.
- Master data management - Get a single, trusted, 360-degree view of data and enable business users to know their data.
- Data preparation - Transform large amounts of raw data into quality, consumable information.
- Data replication - Provide real-time change data capture and synchronization to make data available fast.
- Data integration - Integrate, replicate and virtualize data in real-time to meet data access and delivery needs fast across multiple clouds.
- Data quality - Cleanse and manage data while making it available across your entire organization.

Read more about DataOps, DataOps is not DevOps for data and DataOps principles. The Future of Data Management.

The purpose of data governance is to provide tangible answers to how a company can determine and prioritize the financial benefits of data while mitigating the business risks of poor data. Data governance requires determining what data can be used in what scenarios – which requires determining exactly what acceptable data is: what is data, where is it collected and used, how accurate must it be, which rules must it follow, who is involved in various parts of data?

According to the DAMA-DMBOK: Data Management Body of Knowledge, 2nd edition: »Data governance is the exercise of authority and control (planning, monitoring, and enforcement) over the management of data assets.« This definition focuses on authority and control over data assets. The important distinction between data governance and data management is that the DG ensures that data is managed (oversight) and DM ensures the actual managing of data to achieve goals (execution), in short that DM ensures that data is compliant and ensures compliance.

According to Seiner: »Data governance is the formalization of behavior around the definition, production, and usage of data to manage risk and improve quality and usability of selected data.« This definition focuses on formalizing behavior and holding people accountable. [Robert S. Seiner. Non-Invasive Data Governance: The Path of Least Resistance and Greatest Success.]

If data management is the logistics of data,
then data governance is the strategy of data.

Data management ensures data is compliant.
Data governance ensures data is managed.
Data as a product ensures a better data-driven decision.

The traditional, centralized, data-first approach to data governance represents a top-down, defensive strategy focused on enterprise risk mitigation at the expense of the true needs of staff who work with data. The Data Governance Office develops policies in a silo and promulgates them to the organization, adding to the burden of obligations on data users throughout the enterprise who have little idea of how to meet the new and unexpected responsibilities.

The modern, agile, people-first approach to data governance focuses on providing support to people who work with data. People are empowered but nonetheless, expected to contribute to the repository of knowledge about the data and follow guidelines, rather than rigid, prescriptive procedures.

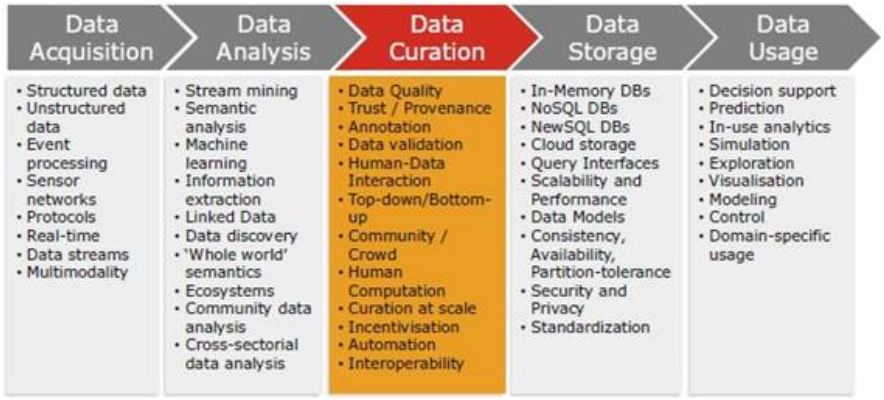
Governance can be divided into three maintainings:

- Business Governance maintaining the requirement, content, quality, and understanding of core business assets and their supporting data.
- Compliance Governance maintaining the policies, procedures, classifications, assessments, and audit standards required for regulatory reporting.
- Technical Governance maintaining the content and quality of extended data attributes and data sources in the enterprise data landscape.

Policies describe what to do and what not do to, guide the way people make decisions. A policy established and carried out by the government goes through several stages from inception to conclusion.
Procedures describe how to do for accomplish and completing a task or process. Procedures are the operations to express policies.
Community of practice (CoP) is a group of people who share a concern or a passion for something they do and learn how to do it better as they interact regularly.

Example of a model of organization where data governance fits as a decision making layer between the executive area and the more operational data management.

A little wider term is **Data Enablement** to enable business users to safely use data where data enablement will be fully aligned with the organization's strategic and organizational goals of data curation. The result is curated data. (In danish helbredte data).



All You Need to Know About Data Curation Big Data Curation

Data Mesh is an architectural approach that decentralises data ownership and management. The approach helps you scale, improve data quality, and increase agility in your data team. Illustration of data mesh Video about data mesh

1.19. Big data

Some properties of big data:

- Volume refers to the amount of data e.g. from GB over TB to PB as data at scale and transactions.
- Velocity refers to the speed of change and data processing for the timeliness of data, data in motion from batch in periodic, frequently, night or daily loads, intra-day e.g. three times a day or nearly real-time with little delay and real-time instantaneously with streaming data with no delay as interactive while an event or a transaction is happening, occurred, registered or took place
- Variety refers to data in many forms and the number of types of data from relational database, image, photo, audio, video, multimedia, sensor, web logs from social network data, smartphone and unstructered text.
- Veracity refers to data uncertainty due to data inconsistency and incompleteness by managing the reliability, trustworthiness, authenticity, origion, repitation, availability, accountability, quality and predictability of inherently imprecise data types.
- Variability to increase in the range of values typical of a large data set and non-standard.
- Value refers to data of many values which addresses the need for valuation of enterprise data e.g. statistical, events, correlations, hypothetical, leverageable.
- Visibility refers to data in the open and being able to handle european General Data Protection Regulation (GDPR) and issues of privacy, security and provenance.
- Visualization refers to the flexible viewing of data.
- Viscosity, Validity, Venue, Vocabulary and Vagueness is other V-words.

A **data lake** can be used for massive quantities of unstructured data and big data with tools that can easily interface with them for analysis for business insights. A datum in a lake has a tag (or a label) to give it a characteristic and to cataloging the data in a data lake. By a tag we can fetch data from the lake without knowing

the physical location like a server url with a folder path. A data lake can contain files on multiple servers on premise in different folders and in the cloud (many nodes), and we only using a tag to finding, fetching, searching, exploring or discovering data.

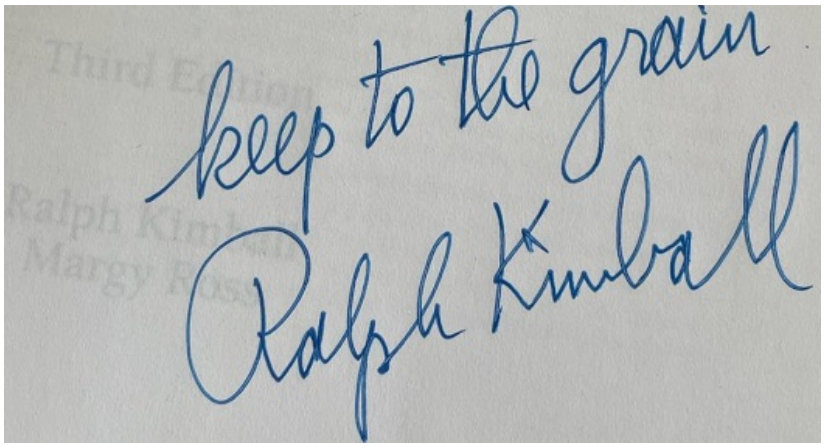
For example, I like to find photos of smiling employees in all albums, I can search for a tag FacialExpression = smiling. A data lake is using ELT (extract, load, and then transform). A tool for a data lake can be like Apache Hadoop or Microsoft Azure. Data Discovery Area is a sandbox for business users. Can use U-SQL to dive in the data lake to fetch the wanted data and do the wanted transformations.

Read about a data lakehouse. SQL on-demand is a query service over the data in your data lake. Azure Synapse Analytics formerly known as SQL DW.

A **NoSQL = Not Only SQL** database means it can use SQL type query language, but usually do not do so. NoSQL database often designed to run on clusters, made by open source and the database does not operate with a fixed schema structure but allow the addition of data without a pre-defined structure. A NoSQL database is characteristic by BASE (Basic Availability, Soft state, Eventually consistent). A ACID system guarantees data consistency after each transaction; a BASE system guarantees data consistency within a reasonable period of time after each transaction. In other words, there is data consistency in the system, just not immediately. This leads on to the Soft State principle. If the data is not consistent at all times, the system must take a temporary data state into account. The sum of both these principles means that data accessibility is given very high priority, even if coincident errors occur in the database system, operating system or hardware. If parts of the database does not work, other parts of the database take over, so that data can always be accessed.

2. Grain (granularity) of dimension and fact

The grain is the level of detail of data stored in a table, both in a dimension table by the hierarchy and in a fact table by the definition of the measurements from a business process event. Importance of granularity is to declare the grain of a fact table to tell what a fact table row represents. Grain declaration for declaring the grain is important to do before identify a measurement and a dimension for »everything we know to be true« about the grain. (In danish granularitet).
A business key has the same granularity and the same semantic meaning across a business organization company.



Declaring the Grain by Ralph Kimball

A grain is defined by one column or composite columns and the columns must contain a value, grain column(s) must not allow null, must have a constraint not null to enforcing it to always contain a value. A grain can be implemented as a constraint unique key (unique nonclustered index), sometimes as a primary key in a table. A table has one grain. A table can never have two different grains, instead it becomes two tables, where the grain columns always have values.

2.1. The grain of a dimension table

The grain of a dimension is determined by its business key, that is either a single column or composite of two or several columns.

A slowly changing dimensions SCD type 0 or type 1 or type 3 dimension table has a business key value as a unique identifier for each row in a dimension table, which makes the dimension grain as the business key. A value in a business key column will be in one row only. There can be a constraint unique key (unique nonclustered index) for business key to ensure uniqueness and no duplicates.

A slowly changing dimensions SCD type 2 or type 7 dimension table has a business key value in multiple rows and a unique identifier for each row will include a time-line metadata column ValidFrom. There can be a constraint unique key (unique nonclustered index) for business key + ValidFrom.

A dimension type 2 or type 7 changes is normally tracked on a:

- **daily grain** with no more than one change per day,
  ValidFrom and ValidTo gets a date, e.g. 2015-05-08.
- **minute-second grain** where many changes could occur on a given day,
  ValidFrom and ValidTo gets a datetimestamp e.g. 2015-05-08 14:25:37.
- **weekly grain** or **monthly grain** when source data is loaded once a week or once a month.

Examples of dimension grain
Store dimension type 1 has grain store and a unique business key in column Code.
Date dimension with hierarchy Year→Quarter→Month→Date has grain daily or date.
Product dimension type 1 with hierarchy Category→Brand→Label has grain product and a unique business key in column Number.
Organization dimension type 1 with hierarchy
Division→Subdivision→Unit→Department→Team
has grain team and a unique business key in column TeamName.
Customer dimension type 2 has a non-unique business key in column CustomerId and a timeline for a CustomerId value has grain CustomerId + ValidFrom as a daily grain.

2.2. The grain of a fact table

Every fact table is defined by the grain of the table.
What is the grain of a fact table?
The grain of a fact table is a business definition of a measurement business process event that makes a fact table row.
The grain is the business definition of what a single fact table row represents.
The grain of a fact table describes exactly what a fact table row represents.
The grain of a fact table can be an individual business process event where a fact table row represents a transaction, a circumstance, an instance, an incident or a state in time.

The grain of a fact table must be declared before choosing dimensions and measurements, measures, metrics, KPI from a business process event.
A fact is expressed in terms of dimensions.

Kimball page 11: »All measurements rows in a fact table must be at the same grain.« Page 39: »Different grains must not be mixed in the same fact table.«
No go to a multi-grained fact table. Read about Grain and Row number in SQL.

To determine the appropriate level of granularity, consider the following factors:
· Business requirements: Identify the most granular level of data needed to answer the business questions and support the desired analytics.
· Query performance: Consider the impact of granularity on query performance. More granular data results in larger fact tables, which can affect query response times. However, this can be mitigated through the use of roll-up dimensions.
· Storage costs: Assess the storage requirements for different levels of granularity. More granular data consumes more storage space.
· Data availability and quality: Evaluate the availability and quality of data at different levels of detail. Granular data may not always be available or reliable.
Source reference.

Examples of fact grain
Sales fact grain is that a single fact table row represents a sale on a purchase date to a customer of a product by an employee in a store.
A customer can buy several different products in a sale, which are represented in several rows in the Sales fact table.
Purchase date is useful for the grain, because the sales fact has no column for a Sale Id or Purchase identification or InvoiceNumber or PurchaseOrderNumber.

Illustration of a Sales fact without a natural Id from a source system

For example in a sql statement to fetch rows that is a grain breakage:

```
SELECT PurchaseDate_key, Customer_key, Product_key, Employee_key, Store_key,
    NumberOfRows = COUNT(*)
FROM Sales_fact
GROUP BY PurchaseDate_key, Customer_key, Product_key, Employee_key, Store_key
HAVING COUNT(*) >= 2
ORDER BY 6 DESC
```

Shipments fact grain is that a single fact table row represents a shipping item in a customer's order for a product.

A customer can order several different products in an order to be shipped, and these products are represented in multiple rows in the Shipments fact table.
Fact column order_id_app is useful for the grain.

Illustration of a Shipments fact with an order_id_app from a source system

For example in a sql statement to fetch rows that is a grain breakage:

```
SELECT order_id_app, product_key, NumberOfRows = COUNT(*)
FROM shipments_fact
GROUP BY order_id_app, product_key
HAVING COUNT(*) >= 2
ORDER BY 3 DESC
```

Cottage covering fact grain is that a single fact table row represents a date for a cottage with coveringstatus as rented, vacant, cleaning, maintenance. Dimensions like Cottage, Landlord and Tenant and a measure like rental price per day.

An individual boarding pass to get on a flight, where the grain is »boarding pass by flight«.

A daily snapshot of the inventory levels for each product in a warehouse, where the grain is »daily by product«.

A monthly snapshot for each bank account, where the grain is »monthly by bank account«.

A relationship tells that a customs declaration can have many versions, and a version can have many goods items. DeclarationGoodsItem fact grain is that a single fact table row represents a goods item within a declaration version in a declaration.

I would not recommend multiple declaration versions in the same fact table, because a gross mass will not be a fully-additive measure. You must first specify a declaration version which is not easy when some declarations have two versions, others have four versions or six versions. Instead the fact table will mark the latest declaration version with a metadata column IsCurrent = 1 (true) and the older versions with IsCurrent = 0 (false) to make it easy to sum up the total gross mass for the latest version of all the declarations.

I prefer to have two fact tables with each their own grain: A DeclarationGoodsItem fact with latest version and current version of the declarations, therefore no need for a metadata column IsCurrent and a gross mass will be a fully-additive measure, and a DeclarationGoodsItemHistory fact with previous declaration versions which I think will not be used in a BI tool, only used to obtain information about previously registered values.

In the DeclarationGoodsItem fact table I will let a row contains the latest process status of the declaration because it is independent of the declaration version and the goods item, therefore all the fact table rows for the same declaration has the same process status.

DeclarationGoodsItem fact grain is that a single fact table row represents a goods item in a declaration for the latest declaration version.

A fact table content must never break with the grain of the fact, because sum up the total gross mass will be incorrect, for example:

Declaration_key	DeclarationVersion	GoodsItemNo	LatestProcessStatus_key	Gross_mass
8052374	2	1	5	400
8052374	2	1	5	400
8190781	1	1	7	178
8190781	1	2	7	162

The first two rows are duplicates, and one reason could be the use of a Left Join instead of a Lookup to fetch a dimension key value for a business key.

The third and fourth rows are for a declaration with two goods items and they have their own Goods Item Number.

A ETL process must guarantee that no grain breakage can occur, otherwise your data quality monitoring should detect it. Or the fact table must have a constraint unique key on Declaration_key + GoodsItemNo to ensure uniqueness and no duplicates so a fact table row presents the latest version of a declaration's goods item.

For example in a sql statement to fetch rows that is a grain breakage:

```
SELECT Declaration_key, GoodsItemNo, NumberOfRows = COUNT(*)
FROM fact.DeclarationGoodsItem
GROUP BY Declaration_key, GoodsItemNo
```

```
HAVING COUNT(*) >= 2
ORDER BY 3 DESC
```

Breaking the grain makes it impossible to analyze individual transactions and an aggregation or a summarization will provide wrong values in a dashboard or report.

Atomic grain and rolled-up summary grain
The lowest grain, called atomic grain, keep all relevant data from source systems and is therefore the most flexible approach but also take most storage space and easy can cost high query performance. Or most detailed grain available.

To improve query performances the grain can be lifted up to higher level, called rolled-up summary grain, where data will be aggregated and summarized and become an Aggregate fact or a Summarized fact to reduction in grain or increased grain (greater dimensionality) than the atomic facts on their own. It is a derived fact from an atomic fact as a base-level fact with detail level data to a new fact as an aggregate-level fact with overall level data.

Data can also be divided such as the current year data is in grain daily, the previous year in grain weekly and the older data in grain monthly, because very detailed information is normally not relevant for analysis years back in time. These three grains will make three fact tables with each their own grain, because a fact table can only have one grain for all the fact rows.

The conformed dimensions will usually be granular atomic, because each row in these tables most naturally corresponds to a single description of a customer, a product, a promotion or a day. This makes it quite easy to bring in the associated fact tables at the intersection of all these granular dimensions. In other words, the base-level fact tables in the first-level dimensional model should be at the natural lowest levels of all the constituent dimensions.

Types of fact grain
The granularity of a fact table can be divided into five types of grains, with Kimball calling the first three fundamental grains:

Transaction grain with one row that represents the finest level of detail with one row per transaction together with a transaction datetimestamp like a entry date, post date, sales date or simply a created date, inserted time, registered date or happening date of the row, e.g. a bank transaction. Sometimes there is a column for a TransactionId (transaction identification) or EventId to be a good candidate for a fact grain if there is no counterpart with same Id value. It is a first-level data mart with a base-level fact table.

Periodic grain with one row that represents a group of transactions through summation made over a period of time, a time period or a time span like a hour, a day, a week, a month, a year for a hourly, daily, weekly, monthly or yearly grain including summarized measurements. A date dimension is represented by a day level, a week level or a month level with the first day of the each week or month. Notice, that certain dimensions are not defined when compared to a transaction fact such as dimension TransactionType.

Accumulating grain with one row that represents the entire lifetime of an evolving event with a clear beginning and end, and therefore is constantly updated over time.

Aggregated grain with one row that represents a summarized level and can be with a shrunken dimension.

Temporal snapshots by applying temporal table or system-versioned table theory where each column of data will have its own timeline metadata columns like ValidFrom and ValidTo to tell when the the value in a row and in a column were valid. Temporal snapshot fact table allows to have the equivalent of daily snapshots without really having daily snapshots. It introduces the concept of time intervals into a fact table, allowing to save a lot of space, optimizing performances while allowing a business user to have the logical equivalent of the »picture of the moment« they are interested in, source reference.
Example of a temporal fact and description with examples and a picture.
Bitemporal transactional data captures two distinct time dimensions: Valid time when a fact row is true in the real world and Transaction time when the fact row was recorded or modified in a source system or a data warehouse. Bitemporal modeling is good for financial reporting because it is often desirable to be able to recreate an old report both as it actually looked at the time of creation and as it should have looked given corrections made to the data after its creation. (In danish temporal er tidsmæssig data lagring).

Grain yearly to grain monthly to grain weekly to grain daily we say that each level e.g. daily increase the granularity and increase the number of rows in a fact table.

A retail sales fact table has grain store and like to increase the grain to individual cash register in the stores.

Aggregation is the process of calculating summary data from detail table rows (records) and is a powerful tool for increasing query processing speed in data marts. For example, a sales is a fact with measures like quantity sold and sale amount, and dimensions like product, customer and date of purchase brings a sales in a context. The grain of the sales is limit to a date (like December 23) and a time (like 4:30 pm) and therefore the fact is on Transaction grain. In case we drop the time, the measures would be called TotalQuantitySold and TotalSaleAmount because they are the result of a summation of sales times to a sales date and therefore the fact is on Periodic grain. Also if we decide to summarize the sales date to a weekly or monthly level. In case we decide to grain the customers by aggregate them into segments and don't keep names and addresses, then the fact becomes Accumulating/Aggregated grain. A Product dimension has a three level hierarchy of category name, brand name and product name, and therefore we can say that product has the highest level of detail. When a source system only gives a sales transaction with a brand name and no product name, a Product dimension must fit that grain of fact and take out the product name level so brand name becomes the lowest level that match the fact grain.

Instead of summation of sales times or sales date and loose some important information of time of purchase, we can summarize data into two other fact tables for a weekly level for data from the last year to this year, and a monthly level for older data, because when we go back in time, we don't need to analyze on a daily or weekly level, and by aggregation we save harddisk space and improve the performance of the query because fewer rows in month level need to be summarized to fetch the year level data.

Granularity is a description of a level of detail of data, e.g. the combination of customer and product is a fine level to tell me about sales, and when I add store to the combination the granularity is driven one level lower, and when I add salesperson to the combination the granularity is driven one level lower and I know much more about sales, and when I add a date or better a datetimestamp it will bring the granularity to its lowest level or to the highest level of detail of sales and a sale.

The lowest level of aggregation or the highest level of detail is referred as the grain of the fact table.

The grain of a fact is determined by either the combination of dimensions or the actual transaction level. For aggregate fact or summarized fact the grain is the intersection of its dimensions.

For example, an accumulating snapshot fact table receive data from a school system to capture a student through different stages (register date, enroll date, exit date). A transactional fact table receive data from a school system to capture the student coursework where students are following courses and later finish them or sometimes a student drop a course, a dimension CourseStatus is nice to have to keep track of the students over time with values like Active, Completed, Suspended or Dropped together with a Date dimension to tell when a student began a course and ended the course. Accumulating snapshot fact has flattening multiple rows into columns of different stages (registered, enrolled, exited).

3. Fact

A fact provides a measurement to a dimension with context descriptive data, and a fact row represents an individual business process event as it occurred or took place at that point-in-time.

A fact table represents a business process, a business process-centric fact, and a business process event, e.g. a transaction, a circumstance, an instance, an incident or a state with a business entity will represent a measurement in a fact table with measurements, measures, metrics, KPI.

A fact table contains business element with measurements and columns that contain references to dimension tables with context descriptive data values, so the containing measurements make sense to business users in different business functions and departments in a company or an organization.

A fact table is designed around numerical measurements. When a measurement is taken, a fact row comes into existence. The measurement can be the amount of a sale, the value of a transaction, a running balance at the end of a month, the yield of a manufacturing process run or a laboratory measurement, etc. If we record several numbers at the same time, we can often put them together in the same fact row. We surround a measurement with all the things we know to be true at the precise moment of the measurement.

A fact table can contain fact's detail level data at a base-level or fact's overall level data at an aggregate-level as a derived fact as a consolidated fact, an aggregate fact, a summarized fact, an accumulating snapshot fact that depends of the grain approach.

Conforming fact means making agreements on common business metrics such as key performance indicators (KPI) across separated source systems so that these numbers can be compared mathematically for calculating differences and ratios (Kimball page 446).

It is almost always a mistake to mix disjoint kinds of measurements in the same fact table.

A dimension has a value or member in a column, and values or members in a row.
Read about dimension table in chapter 4.

3.1. Types of columns in a fact table

There are several types of columns in a fact table.
Do not create a big wide fact table as performance would be terrible.
Try to keep a fact table as lean as possible.

Dimension key is a foreign key in a fact table to a dimension table.

A dimension key refers to a dimension table with a column dimension key as a primary key, and a dimension provides a context for a measure with descriptive data values from descriptive data columns in a dimension table.

A dimension key column must not allow null, must have a constraint **not null**, therefore a ETL process will replace or coalesce null in a business key column to an artifact value or inferred member, e.g. -1 for missing.

A dimension key column is represented as an integer or as a date.

Kimball prefers a suffix **_key** for a dimension key column in a fact table as the foreign key for a dimension. An alternative suffix as _dimkey or _dim_key.

For example, Customer_key column in a Sales fact table which is referring to a Customer dimension to a Customer_key column as the primary key.

In a fact table a dimension key column Date_key is a general term, therefore it is better to give it a prefix label and let it play a role, e.g. »purchase« becomes PurchaseDate_key in a fact table for a role-playing date dimension.
I.e. PrimaryResponsible_Employee_key, SecondaryResponsible_Employee_key.

Kimball prefers a column name for a dimension key in a fact table to be independent of the types of slowly changing dimensions, therefore no indicator in prefix or suffix regardless of dimension type 0 – 7.
Read about types of slowly changing dimensions in section 4.3.

When a dimension is a type 1 dimension it provides current values, whereby there is a data loss from an operational system or a source system compared to the data values when a fact was happening, occurred, registered or took place.

When a dimension is a type 2 dimension it provides registered values, whereby there is a no data loss from an operational system or a source system compared to the data values when a fact was happening, occurred, registered or took place.

A dimension key in a fact does not tell whether the dimension is a type 1 or a type 2, but when I join from a fact table dimension key e.g. Customer_key to a dimension view called either DimCustomer_Current or DimCustomer_Registered, I choose which dimension values I want to fetch and see.

Please, use conformed dimensions to explain fact data in a row from a fact table for the business users.

A fact table express a many-to-many relationships between dimensions and is implemented as one-to-many relationships between dimension table and fact table over the dimension key. I never have foreign key constraint on a dimension key in a fact table, because it decrease inserting performance, and I trust a ETL process and no human being will update data or delete data a dimension table or in a fact table.

When a ETL process is going to insert a new row into a fact table, it will carry out a **lookup** on a business key value in a dimension table to fetch a dimension key value for the new fact row. When it is a type 2 dimension or a type 7 dimension, it will carry out a **date range lookup**. Read about lookup in section 4.3. A ETL tool has a lookup function as a component to fetch one row from a dimension table. A sql statement has a Left Outer Join to a dimension table and be ware it can fetch

several rows which is not good for number of rows that will be loaded into a fact table, see a sql in section 6.2.2.

Measurement (measure, metrics, KPI, analysis variable) that contains quantitative numeric fact as a number value from a business process event e.g. quantity of a product, unit price of a product, bankbook balance amount, loan amount in a bank, length of a meeting with a customer, number of seats in a cinema. All measurements in a fact table must have the same grain, read about it in chapter 2.

A measure column can allow null with two meanings, either a value does not exists or a measurement system failed to capture a value.

A measure column is represented as an integer, decimal, numeric, float, real.

A fraction (e.g. a percentage) and ratio based on a dimension slice will be calculated »on the fly« in a BI tool when a fact table contains columns for a numerator and a denominator with good naming. (In danish en brøk med tæller og nævner).

Sometimes a cube or a BI tool can't perform a count of rows, therefore I add a measure with two integer values 0 and 1 to perform a sum, e.g. ExceededCase or Effect of an effort.

Please, use a conformed measurement to explain a measure and a metric in a fact table row for the business users.

Degenerate dimension contains an operational system transaction identifier that has no association to a dimension table in a data mart area. The column is labelled DD and is called »a dimension key without a corresponding dimension« because there is no descriptive data values connected to a degenerate dimension column in a fact table which would give an »empty« dimension table. A degenerate dimension column in a fact table is a dimension key without an associated dimension table to join to. [Degenerate refers to something that is: declined from the standard norm or is mathematically simpler.]

A degenerate dimension column can allow null when a value does not exists.

A degenerate dimension column is represented as an integer or text.

A degenerate dimension happens when a dimension column is stored as part of a fact table, and not in a separate dimension table. Degenerate dimensions commonly occur when the fact table's grain is a single transaction or transaction line and is assigned by an operational business process, e.g. a credit card transaction.

Keep operational control numbers such as order numbers, invoice numbers in fact table and use it as degenerate dimension. In a purchase order fact table column PurchaseOrderReference is labelled DD or (DD).

Degenerate dimensions often play an integral role in the fact table's primary key. Instead of combining all dimension keys in a fact table for primary key, a degenerate dimension column can guarantees the uniqueness of a fact table row.

For example, the grain of an Order fact table is an order line item number because an order often consists of several products and they are connected to an order number, therefore the Order fact table has two degenerate dimension columns called OrderNumber and OrderLineItemNumber with data from an order operational system. The OrderNumber column is a grouping key that »glue« multiple rows together for all the products in an order to sum up the total order amount or to count the total number of products. The Order fact table has other dimension keys for OrderDate_key to a Date dimension and for OrderType_key to an Ordertype dimension with a Code, a Designation, a Name and a Description and a division of types into a hierarchy.

Other examples of degenerate dimension columns in fact tables: Invoice number, Ticket number, Transaction number, Voucher number, Receipt number, Version number, CaseId, EventId, MovementId or TransactionId (transaction identification).

For example, an incurance policy has often a risk grade, therefore I will create a natural business dimension called Policy with the business key PolicyNumber and data element RiskGrade and other descriptive data values. It will be a type 2 dimension to keep old values of RiskGrade. Dimension key will be Policy_key and it will be placed in one or several fact tables. PolicyNumber will not be a degenerate dimension in a fact table.

For example, European customs authority is issuing a MRN Movement Reference Number, now called Master Reference Number that is unique for each shipment. A customs declaration with a MRN is used in several business processes like Shipment, Inspection and is used to track the status of a shipment and for documen-

tation and communication with customs authorities particularly during import and export.
MRN consists of 18 characters (letters and numbers), where the first two characters are the year in which the goods are shipped, e.g. 24 for 2024, and the next two characters are the two-letter country code of the country where the shipment originated, e.g. DE for Germany and DK for Denmark. Since MRN is **data-bearing**, I will create a dimension called Declaration with the business key MRN and derived columns MRN_Year and MRN_CountryCode plus other relevant information about a declaration such as creation date, latest version number and date, latest status of process, UCR Unique Consignment Reference, DeclarationType (A1, A2, H1, H7) and other EUCDM data. It will be a type 1 dimension, and it is very rare for these data to change, only due to correction which gives a new version number and it will overwrite a row.
I expect many millions of rows in the Declaration dimension but it is fine for degenerate dimension compared to a regular dimension.
I will not make outrigger dimensions from the Declaration dimension, instead I will add to fact tables a DeclarationType dimension with different hierarchies and descriptions.

Degenerate dimension in a fact table must be non-data-bearing, I think.

Kimball Design Tip #30 »Put Your Fact Tables On A Diet« has guidelines to keep a fact table as lean as possible, and one of the advice is: »Take all text fields out of the fact table. Make them dimensions. Especially comment fields.«

Kimball page 94 mentions a header entity and to create a normal dimension and no longer have a degenerate dimension. If there are several transaction-grain columns in a fact table that do not belong to any dimension then create a normal dimension.

A dimension with a grain same as a fact, like a Transaction dimension, is questionable because a join between fact and dimension costs performance, but this dimension is typically only used in an atomic drill-through operation, so the cost of join is minimal. A good debate in Kimball Design Tip #86.

Tag contains a text, a flag, an indicator, a datetime or a number that can not be summarized. A tag column in a fact table is a candidate for a degenerate dimension or later a real dimension with extra descriptive data columns. Kimball page 48, 82, 254-256 and 301. A tag column can allow null.

Sometimes I see a flag column with data type of boolean/bit/char(3) with two values True (1, Yes) and False (0, No) as a flag indicator and the column may even be null, and I ask a data modeler why not use a Yes/No dimension with data values »Yes« and »No« and »No answer« for null.

How to name a boolean (bit) value for a conversation due to absence, e.g.:
IsConversationDueToAbsence or ConversationDueToAbsenceFlag or
ConversationDueToAbsence_YesNo_key to indicate a role-playing Yes/No dimension in an EmployeeAbsence fact table.

Sometimes I see a column with data values as a datetime in seconds or fractional seconds in milliseconds, and I ask the data modeler if millisecond really is necessary, and split the column into a Date dimension and a Time dimension with grain as minute or second. The data modeler says the datetime column is important to compute very precise time intervals, and Kimball ETL Toolkit page 173 and Kimball Design Tip #51 recommends an embedded datetime in a fact table and says we should think of it as a special kind of fact.

Some data modelers say that a fact table should be lean with measures and dimension keys. It will make queries faster and keep the data model actually usable instead of a bloated mess nobody can navigate. More reading in Kimball page 179.

An illustration of a star schema with a Junk dimension named Order_Info to store code, flag and indicator columns to get a **lean Order fact** instead of having a big wide Order fact table.

Primary key to uniquely identify fact rows to be updatable if data is changed or deleted in a source system and to avoid duplicate rows. But it conflicts with non-volatile rule! A true transactional fact accepts a counterpart row to handle changed, corrected or cancelled data in a source system. Therefore a degenerate dimension column TransactionId can’t be a primary key but can be in combination with a TransactionDatetime.

Consider a primary key for each fact and don’t worry when your fact table does not have a »natural« primary key. Sometimes dimension keys can be involved in a composite primary key and the grain of the fact is important here. The Achilles heel of a composite primary key of dimensions is, when several rows have missing or unknown members or values in their dimensions and the date dimension is the same in the rows, we will have a primary key violation.

The primary key constraint can be non-clustered to get better insertion performance where data is loaded in batch.

I prefer a primary key in a fact table to be an artificial auto-generated unique sequence number or an identity column as a surrogate key, often clustered primary key, e.g. in a FactSales table a primary key column name could be Sales_Id or FactSales_Id or FactSales_key.

A primary key is important in case of linking to another fact table for drill-across or is useable for parent-child relationship link between fact rows. For example, an order row has a reference to a cancel row of the order and the cancel row has a parent reference to the previous order row, a link column pointing to the surrogate identity column. Here you have to consider a clustered primary key.

Business key (**Durable key** from a key mapping table) for each dimension along with dimension keys when a fact table is at lowest grain, and a data warehouse solution has no Archive area, Persistent staging area, Landing zone area or Operational Data Store and is not able to reload raw data or source data, whereby a fact table is the only storage of data. In case of a reload of a dimension, which will result in new values in the dimension key, a business key or a durable key in a fact table will be used to update the dimension key column with the new values, e.g. for type 2 dimension together with a date range lookup, it is good to keep a transaction date column in a fact table.
Durable key is a column in a fact table for a type 7 dimension.

Otherwise a business key column TransactionId is a degenerate dimension.

Technical examples:
- TransactionId or EventId as a data lineage back to the source data.
- TransactionDate or TransactionDatetime (in danish Transaktionstidspunkt) (or EntryDatetime) can be divided into two dimensions for Date and Time, and the Date column is used for partitioning of a table to a partitioned fact table.
- Replacement reference column from a new fact row back to a soft deleted row.

Audit and metadata columns for housekeeping and data lineage
Section 6.3 describes several columns: InsertTime, UpdateTime, InsertExecutionLogId, UpdateExecutionLogId, IsCurrent, IsDeleted, BeginAt, EndAt, DeletedTime (ModifiedTime). BeginAt and EndAt dates in a fact table to represent a timeline period where a measurement was valid or was effective, e.g. a BalanceAmount, read more in section 6.4.

For data lineage a fact table will include audit columns from the Archive area as RecordSource, ArcRecordId and ArcGlobalId for traceability back to the archive.

3.2. Types of measurements, measures, metrics
There are three types of measures in a fact table.

Fully-Additive measure - summable across any dimension
A fact table has numerical measures that can be summed up for all of the dimensions in the fact table, so the measure columns data type is a number. A Sales fact is a good example for additive fact with measures like Quantity sold and SaleAmount. In case of a transaction data set to a fact table refer to a measure column which value is empty, null or nullable, use the default value 0 because this won't bother aggregation like summation of a duration metric.
Each measure must have its metrics.

When it is a monetary measure, it may have a currency column. If it is an unit of measure (UOM) it may have a column to explain the kind of units used like centimeters, litres, cubic metres, etc. Fact can have a calculated measure or a derived measure based on existing measures and constants e.g.:
Profit or Surplus = Revenue – Costs.

Semi-Additive measure - summable across some dimensions
A fact table has measures that can be summed up for some of the dimensions in the fact table and not for other dimensions. For example, a daily balance measure can be summed up through the customers dimension but not through the date dimension. Inventory levels cannot be summed across time periods.

Non-Additive measure - not summable for any dimension
A fact table has measures that cannot be summed up for any of the dimensions in the fact table. For example, a room temperature fact is non-additive and summing the temperature across different times of the day produces a totally non-meaningful number. However, if we do an average of several temperatures during the day, we can produce the average temperature for the day, which is a meaningful number. Sum of a measure called DaysForOrderToCompletion in a FactOrder is mea-

ningless but finding a minimum, maximum and average values are meaningful to planning of production.

Other examples is a percentage and a ratio are non-additive measures. A fact table with columns for transaction number such as an order number, an invoice number or a voucher number can't be summing up. An order fact with measures like Unit price for a single product makes no sence to summarize, but the derived column Sale amount = Unit price x Quantity will be summarized and becomes a fully-additive column, also called a calculated measure.

Trend, Stock and Ranking can't be added and in general all calculations on one specific intersection of the dimension. Year-to-Date ytd measure can't be summed up. Count of rows is normally used.

Unit price can be stored in a Product dimension as a current standard list unit price and still keep Unit price in Sales fact together with a Discount (%) when a purchase was happening, occurred, registered or took place. Back in 2016 I had a customer that used two kinds of discounts when a product had a bundle option and was showned in an invoice line with description »Discounted Bundle«:

Unit price = List price - Unit discount amount.
Line amount = Unit price x Quantity – (Unit price x Quantity x Line discount percentage / 100).

Some business users used the term List price other users used Catalog price.
When there was a promotion going on, the Unit price had a different calculation:

Unit price = List price - Proration amount (also called deduction amount).

There was also a Line discount amount and an Order discount amount. I added all the columns into the Sales fact.

Mostly a fact table describes what has happened over a period of time and has fully-additive measures.

3.3. Types of fact tables

There are several types of fact tables.

Conformed fact
A conformed fact table has measures in multiple fact tables and must use the same common business rule such as naming, definition, calculation method and unit, also across data marts.

When a measure e.g. Revenue is used in multiple fact tables with different calculations and meanings, it is best to use different column names because theses facts are not conformed.

Factless fact
In dimensional modeling, a factless fact table connects a dimension table to a dimension table. Factless fact table is used to resolve many-to-many relationship between a dimension table and a dimension table. Factless fact is used to represent a many-to-many relationship among dimensions. (Kimball Design Tip #50).

A factless fact is to combine data that have a many-to-many relationship between a dimension and a dimension at the conceptual level and break it down at the logical level to a factless fact table with two one-to-many relationships to two dimension tables.

Factless fact relates dimensions, when dimensions have a relationship according to a business process event.
Bridge relates fact and dimension, and take care of a multi-valued column.

A factless fact table contains no measures, no metrics and no numeric additive values as a normal fact table do, we have a measureless fact that records a business process event.

For example, a factless fact table to keep track of relationships of other dimensions and tracking a business state, e.g. a relationship between an employee and a department from an Employee dimension and a Department dimension as a current association.

A factless fact becomes a **timespan factless fact** by including timeline metadata columns BeginAt and EndAt (Effective and Expiration or ValidFrom and ValidTo) when a relationship among dimensions have a time frame, a time period or a time span to tell when a fact row was valid or was effective.

For example, a timespan factless fact table to handle these business process rules:

- over time an employee may have multiple departments
- at same time an employee may have multiple departments

metadata columns BeginAt and EndAt tells when an employee was working in a department, or currently is working with EndAt set to the »end of time«. When an employee resigns, EndAt is set. Some years later the same employee is rehired,

there will be added a new row in the timespan factless fact table with a new date in BeginAt.

For example, a factless fact table with only columns for employee, date, time and event of work like »workstart«, »workstop« and »workstopofsickness« and no columns for a measure. You can get the number of employees working over a period by a »select count(distinct Employee_key)« or by a distinct row-count calculated measure in your OLAP cube, DistinctCount, when Employee is a Kimball type 1 dimension. For a factless fact you will normally count the number of rows, row count or counting rows and call it »Number of <a name><«.

Sometimes a factless fact table has a value column called Count with only one value as 1 used in a data access tool to sum up and get the number of rows. In case a fact grain is weekly and a week is missing, it can be inserted to have all weeks complete and here will column Count gets the value 0.

Factless fact is for registration of event, process or assignment e.g. attendence take place in a school class with dimensions for Student, Class, Room and Professor. If we add a measure column for Attendence with 1 or 0 per date per student it is not a factless fact anymore.

Capture a relationship in the fact
To be a column in a dimension or to be its own dimension and used in a fact is a good question. Kimball's financial services example starts with an Account dimension including data of products and branches but he choose to remove these descriptive data columns to form independent dimensions of Product and Branch and use them in a fact together with the Account dimension. Therefore the fact capture a relationship among accounts, products and branches. The account-product-branch relationship is a factual relationship, or a delegation **factless fact** table.

Kimball said: »Demoting the correlations between dimensions into a fact table.« I like to add with »capture a relationship in the fact« it keeps the registered relationship at the time of a business process event, a transaction, a circumstance, an instance, an incident or a state occurred and was entered into the fact table.

An example is a retail store where Store is both a dimension and a fact by action:
- A Store dimension has a StoreCode as a business key and Name, Address, Region, StoreType and StoreStatus.
- A StoreCampaign fact has Store, Campaign, FirstDate, LastDate dimensions.
- A StoreObjective fact has Store, First date of a quarter dimensions and measures for Turnover and Revenue.
- A StoreSurveyQuestionnaire fact has Store, ResponseDate, Customer and Questionnaire dimensions with measure for Answerscore between 1 and 10.

Another example is an operational system that contains current relationship of which buildings house which departments are located. Over time a department will relocate to another building. When we capture a relationship in the fact it will include history of previous locations with a time period or time span of begin date and end date. See more later at Timespan fact as part of Slowly Changing Facts.

An example about could incidents of vehicle accidents be in an Accident fact table with the granularity as per the natural grain for one accident with dimensions for DateTime, Location and Accident with AccidentID and Police Report Number. There can be several vehicles and casualties involved in an accident which leads to bridge tables and group dimensions. An alternative is a lower granularity than one accident, for example into two fact tables that capture two relationships.
* Accident_Vehicle fact with grain »vehicles damaged during an accident« where one accident has several vehicles in rows with dimensions for DateTime, Location, Accident, Vehicle with inferred members for »No Vehicle« and »Unknown« vehicle and Vehicle DamageType. There can be several damages, but I assume there is a superior type to cover them all. A participant roll-playing dimension for a Driver and an Owner of a vehicle.
* Accident_Participant_Injury fact with grain »injuries to casualties sustained during an accident« where one accident and one casualty has several injuries in rows with dimensions for DateTime, Location, Accident, Participant and Injury with inferred member »No injury« to include those who were not injured.

An example is a retailer SuperChemi belongs to a chain Ethane at date of sale e.g. 2013-05-17 and the fact table has two columns for dimensions to Retailer and Chain. In 2015 the retailer SuperChemi changes to another chain Propane but we still keep the registered relationship back in 2013 and 2014 in the fact. When a chain is not a part of the fact table and we in year 2016 like to find sales of retailers for a specific chain e.g. Propane, we will use the current relationship between dimension Retailer and dimension Chain as a snowflake dimension, meaning that SuperChemi belongs to Propane, and when we summarize sales per year since 2010, chain Propane will include the sales of SuperChemi in 2013 even though the retailer belonged to Ethane at that time.

When we capture a relationship in the fact, we will use a dimension key from a type 1 dimension and a type 2 dimension, or a durable key from a type 2 dimension.

Transactional fact or Transaction fact or Transaction-grained fact table
A transactional fact table or a transaction fact table describes a business process, an event or operation that occurred at a point-in-time in a source system e.g. an invoice with line items. The row has a date e.g. a transaction date, an invoice date, an entry date or a post date or a datetimestamp to represent the point-in-time with other lowest-level data. A date range lookup for a type 2 dimension key value will use a transaction datestamp or a datetimestamp.

Sometimes there is a column for a TransactionId (transaction identification) or EventId to be a good candidate for a fact grain if there is no counterpart with same Id value. It is a first-level data mart with a base-level fact table.

If data needs to be changed, corrected or cancelled in a source system, then a data warehouse needs to make a counterpart row in the fact with a new time stamping or a time span. Source data examples, booking a hotel room, order a car in a special color, buy a lottery ticket with my lucky numbers, receive an invoice, put and pick items in stock. (In danish lægge varer på lager og plukke varer fra lager).

In the dimensional modeling methodology I try to interpret some data as transactional data or transactions. Some data modelers do not like separate facts for each transaction type but build a single blended fact with a transaction type dimension and a mix of other dimensions can make N/A dimensions.

I have seen a blended fact called FactEvent which I think is a poor and non-descript name of a fact table, and the date dimension gets a generalized name. I prefer multiple facts where the date dimension is role-playing for e.g. ordered date, purchase date, shipped date, delivered date and receipt date.

I have seen a big wide fact called FactCase which I wish was divided into:
FactCaseStartup, FactCaseCollectData, FactCaseProcess, FactCaseResult and each fact has different dates as role-playing.

Snapshot fact versus Transactional fact
Snapshot fact represents a state, e.g. my bank balance right now is $500, and tomorrow it is $600 because I will deposit $100, therefore the latest date of balance contains the current balance. It is easy to see my balance ten days ago by look it up at the SnapshotDate. It is like a picture from the past.

Transactional fact contains all my deposits and all my withdrawals with a date and an amount to records the »happening of an action«. In any point of time (normally today) I can calculate my balance, or the fact contains a running balance measure per transaction/per fact row.

Snapshot fact calculates the balance of my account at the end of each day because there can have been many deposits and withdrawals within a day. This snapshot fact can then be easily used to calculate the average daily balance for interest or fees. The balance in the snapshot fact is Non-Additive, is not able to add or sum up together into a meaningful metric, but able to use aggregate functions like Avg, Min and Max.

A fact contains order lines and a ETL process will update an order line with changed data, insert a new order line or delete an existing order line.

A counterpart fact is a kind of a transactional fact that store the difference between the original order line and the new order line in a net change column.

A periodic snapshot fact represents an order and its order lines for a specific period.

A discrepancy snapshot fact will store a new complete image of the order lines when one of them has changed.

A data entry action snapshot fact will store a new order line as a logbook with three actions Added, Changed, Removed.

Periodic snapshot fact
A periodic snapshot fact table describes a state or a status of things in a particular instance of time or a one point-in-time, and usually includes more semi-additive and non-additive measures. A periodic snapshot fact represents states or statuses over time where a snapshot ensures that data is never changed.

It is a table with frozen in-time data, meaning a row will never be changed/modified/deleted (is unchanged) because the row can have been used in a report like a monthly or annual report and later it is a must to be able to create the same raport with the exact same data.

The periodic snapshot fact table will never be done empty or updated, therefore it is a true incremental load. A table has a snapshot column like a Day (yyyymmdd) for daily basis, Week (yyyyww) for weekly basis, Month (yyyymm) for a monthly basis or a Year (yyyy) for annual basis to fetch the wanted data as a day slice, week slice, month slice or a year slice. When a day, a week, a month or a year is over and data is ready, data will be loaded.

Sometimes data for the current month is also loaded every day for a current-month-to-date, therefore the current month will be updated until it is over, finish and can be closed. Measures can be a balance in account or inventory level of products in stock and so on. Key values for dimensions is found at the end of the period.

The grain of a periodic snapshot fact is a period such as daily, weekly, monthly or annual partitioning for making a faster query performance when searching for a specific day, week, month or year. In case there is no data for a specific month it will be nice to insert an artificial row with »missing« value of the dimensions and 0 in the measures. (Periodic snapshot in danish »en skive data«, fastfrysning).

It is common that fact data needs a correction from a source system which can cause a redelivery of a snapshot e.g. a specific month that needs a reload in a periodic snapshot fact. When a fact table is implemented as partitioned table, it is easy to truncate the related partition and load it again (Kimball page 517).

A ETL process can use a staging table and perform a switching in to a fact table related partition. A redelivery can contain a date of redelivery or versionnumber in case of several versions of the same delivery, sometimes called a generation. It is called **incremental refresh** when a ETL process does a dynamically replace or reload of a partition of a table by empty the partition and load data with a higher date of redelivery or higher versionnumber into the partition, because it is refreshing an existing delivery in a snapshot.

Example of Monthly snapshot fact for HR department with dimension keys for a month and employee and measures for salary paid in the month, vacation earned and vacation taken.

Accumulating snapshot fact or Accumulative snapshot fact or Accumulated
An accumulating snapshot fact table describes the activity of a business process that has a clear beginning and end, and is stored in one row for the entire lifetime of the evolving business process events that is representing the entire history of something up to the present.

An accumulating snapshot fact table stores measurements that accumulate across a well-defined period or workflow. It often records the state of a business process at distinct stages or milestones, which might take days, weeks, or even months to complete. The fact table has multiple date columns to represent states, stages or milestones in a business process with different values stored as is a snapshot in a period of time.

Each stage of a business process lifecycle has its own columns, e.g. milestones of a hospitalization, or steps of manufacturing a product, or track of the status of an invoice with statuses such as open, closed, reminder, paid, unpaid, settlement.

An accumulating snapshot fact can have multiple status columns with different names to represent milestones in a business process with different values, e.g. dates, and the columns will be filled out gradually that reflect the completion of events in a lifecycle process to representing the entire history.

The grain of an accumulating snapshot fact is one row per business process with several roles of a date dimension from related business process events to the business process, and each fact table row is updated multiple times over the lifetime of the business process. (Kimball Design Tip #140).

Using the accumulating snapshot fact technique a business user can see a processing pipeline in a simple format, regardless of which source systems data is coming from and which application tool they are using.

An accumulating snapshot fact can be placed in a consolidated mart.

A FactOrder has a grain of an order for one product and for one customer per day where order has three stages for a product with three dates of movements as ordered, shipped and delivered, that become three role-playing date dimension keys in an accumulating snapshot fact table with dimension key column names for each stage of a lifecycle of an order, e.g.:

OrderedDate_key, ShippedDate_key, DeliveredDate_key.

A new FactOrder table row will start with date key value as 0001-01-01 or -1 for »hasn't happened yet« or »be available later«.
Over time an accumulating snapshot fact row will be revisited and updated multiple times while an order moves through the order fulfillment pipeline as more information is collected from every stage of the lifecycle of an order. Read more in Kimball page 194-196.

Other dimension keys: Customer_key and Product_key. Measure is Quantity.

Example of Transactional fact and Accumulating snapshot fact

Transactional fact grain is one row per order and per stage.
Measure Quantity is Semi-Additive per stage.

Order No	Customer _key	Product _key	Stage_key	Date_key	Quantity
351	52	12	1 Ordered	2020-01-05	2
351	52	12	2 Shipped	2020-01-06	2
351	52	12	3 Delivered	2020-01-07	2
352	37	7	1 Ordered	2020-01-05	5
352	37	7	2 Shipped	2020-01-07	5
353	16	9	1 Ordered	2020-01-08	1
354	52	12	1 Ordered	2020-01-08	2
354	52	12	2 Shipped	2020-01-08	2
354	52	12	3 Delivered	2020-01-09	2

Accumulating snapshot fact grain is one row per order with date of stages as columns. The three stages has been pivoted to three date of stages.

When an order has a new date of status, the same fact row will be revisited and updated by overwrite the default date key value 0001-01-01 for »hasn't happened yet« or »be available later«. Measure Quantity is Fully-Additive.
Here is a snaphot from 2020-01-09 at 11:45 am.

Order No	Customer _key	Product _key	OrderedDate _key	ShippedDate _key	DeliveredDate _key	Quantity
351	52	12	2020-01-05	2020-01-06	2020-01-07	2
352	37	7	2020-01-05	2020-01-07	0001-01-01	5
353	16	9	2020-01-08	0001-01-01	0001-01-01	1
354	52	12	2020-01-08	2020-01-08	2020-01-09	2

When order no 352 has been delivered at 2020-01-10, the fact row will be revisited to update column DeliveredDate_key with 2020-01-10.
When order no 353 has been shipped at 2020-01-09 afternoon, the fact row will be revisited to update column ShippedDate_key with 2020-01-09.
Here is a snaphot from 2020-01-10 at 09:57 am.

Order No	Customer _key	Product _key	OrderedDate _key	ShippedDate _key	DeliveredDate _key	Quantity
351	52	12	2020-01-05	2020-01-06	2020-01-07	2
352	37	7	2020-01-05	2020-01-07	**2020-01-10**	5
353	16	9	2020-01-08	**2020-01-09**	0001-01-01	1
354	52	12	2020-01-08	2020-01-08	2020-01-09	2

An accumulating snapshot fact per order with date of stage as three role-playing date dimension keys, can for a report become one date column:

Date	QuantityOfOrdered	QuantityOfShipped	QuantityOfDelivered
0001-01-01			1
2020-01-05	7		
2020-01-06		2	
2020-01-07		5	2
2020-01-08	3	2	
2020-01-09		1	2
2020-01-10			5
Sum	10	10	10

A tabular model creates an Userrelationship in DAX for the Date_key columns to a Date dimension to get one Date column, and use a DAX Switch to handle multiple userrelationships for each role-playing dimension key in a fact table to the same dimension table.

Example of Transactional fact and Periodic and Accumulating snapshot fact

Transactional fact grain is one row per day/hour/minute per account per deposits and withdrawals. Stores data of the most detailed level.
Measure In/Out_amount is Fully-Additive.

Date_key	Time_key	Account_key	Deposit_Withdraw_amount
2020-01-04	1107	638	+95
2020-02-17	1045	638	-25
2020-04-23	1521	638	+12
2020-04-30	0935	638	-7
2020-04-30	1430	638	+1

Periodic snapshot fact grain is one row per end of day per account to reflect the balance for that period. Stores data that is a snapshot in a period of time.
Measure EndOfDay_Balance_amount is Semi-Additive per day.

EndOfDay_Date_key	Account_key	EndOfDay_Balance_amount
2020-01-04	638	+95
2020-02-17	638	+70
2020-04-23	638	+82
2020-04-30	638	+76

Periodic snapshot fact grain is one row per end of month per account to reflect the balance for that period. Stores data that is a snapshot in a period of time.
Measure EndOfMonth_Balance_amount is Semi-Additive per month.

EndOfMonth_Date_key	Account_key	EndOfMonth_Balance_amount
2020-01-31	638	+95
2020-02-29	638	+70
2020-03-31	638	+70
2020-04-30	638	+76

Accumulating snapshot fact grain is one row per account per year with month of balances as columns. The twelve EndOfMonth_Balance has been pivoted to twelve month of balances. Measures Jan to Dec is Fully-Additive.
Here is a snaphot from 2020-05-01 at 08:07 am.

Account_key	Year	Jan	Feb	Mar	Apr	May	Jun	…	Dec
638	2020	+95	+70	+70	+76				

When month of May in year 2020 has a final balance amount, the fact row will be revisited to update column May with a balance amount.

Example of Accumulating snapshot fact for account data
Account basic descriptive data values exist in an Account dimension with Account number (10 characters) as business key and Account_key as dimension key (integer value). **Lifecycle data** exist in an accumulating snapshot fact with grain of one row per account called Account Lifecycle fact including columns for the important and common questions about an account:

- Open Date key
- Close Date key
- Current State key
- Product key
- Customer key for one customer or Customer Group key for two customers
- Current balance amount

Example of Accumulating snapshot fact for payment data
An accumulating snapshot fact table where a row is a summarize of measurement events occurring at predictable steps between the beginning and the end of a process. For example, a source system for current payments from customers where some pay several times over a month, and the first payment becomes a new row in fact table with date of payment in columns BeginDate and EndDate and the amount in PaymentAmount column. The next ETL process will do a summarize of payment per customer from the BeginDate to current date or end-of-month date, and then update the fact row with same BeginDate with the new summarized payment and new EndDate, so a fact row will be revisited and updated multiple times over the »life« of the process (hence the name accumulating snapshot).

Kimball Design Tip #37 has a big example of an accumulating snapshot fact for a student admissions pipeline business process with several business process events.

Timespan fact or history fact or historical fact or Status fact
A timespan fact table or a history fact table or a historical fact table is used for a source system that is regularly updatable meaning that a source system changes and overwrites its values. To capture a continuous time span when the fact row was effective, the fact table will act as SCD type 2 dimension with BeginAt and EndAt columns to keep historical data and to represents the span of time when the fact row was the »current truth«, and with a query it is easy to fetch it at a given point-in-time. Loading of data will be based on incremental load or delta load. It is called Slowly Changing Facts, read more in section 6.4.

Counterpart fact (negating fact)

A counterpart fact table is used to store the difference between the original order line and the new order line in a net change column, is a counterpart fact kind of a transactional fact where data is stored transactionally. When a change occurs, insert a new fact row that reverses the previous row (reverse the sign on all measures, same dimensions), and insert a new fact row that represent the new version. All fact rows would be timestamped as to when they were inserted. Use the timestamp to recreate reports as of a particular point in time.
I label it Slowly Changing Facts because a source system is changing fact measure value without keeping the old value as a historical transaction, read more in section 6.4.3.

Timespan accumulating snapshot fact or State oriented fact or Status fact
A timespan accumulating snapshot fact table or a state oriented fact table where a fact is not a singular event in time but consists of multiple states or events that occur over time so each row represents a state of an object during a period of time in which the state didn't change. It is similar to a type 2 dimension with snapshot BeginAt date, EndAt date and a current indicator/active flag where the row formerly-known-as-current will be revisited and updated. A state is often a status.

The use of stages points towards an accumulating snapshot fact, and the use of states points towards a periodic snapshot fact to snap a picture of a business process.

One solution to the historical pipeline tracking requirement is to combine the accumulating snapshot with a periodic snapshot: snap a picture of the pipeline at a regular interval. The combination is called timespan accumulating snapshot. (Kimball Design Tip #145), and see more later at Timespan fact.

Depending on the frequency and volume of changes, you may consider a »current only« version of the fact to improve performance of what will probably be a majority of the queries. Then you can use the fact to drive the loading of selected events in an accumulating snapshot fact.

Active fact together with historical fact
You can have an active fact with current state or version of a row and another historical fact with previous states and versions of rows together with timeline metadata columns to represent a time span. If an active fact row change with different values from a source system, typical rows in a staging table with business keys and dimension keys and measures that you have compared with rows in an active fact, then you move a row from the active fact to a historical fact where you update timeline metadata columns, and afterwards you insert a new state or version of a row from the staging table into the active fact. No IsCurrent in fact tables and for sure, the business users will query an active fact more than 90%. The dimensions in an active fact will still handle history of rows that have not changed for years, because the dimensions in an active fact will be of type 2 dimensions.

Allocated fact
An allocated fact table is created by allocate the higher-level facts to a more detailed level facts. High level fact versus detail level fact.
From Header fact to Detail fact or from Header fact to Line fact.
For example, the Order header/parent dimensions and measures are allocated down to the Order detailed/child line item fact table. It is called allocation.
Example of header-detail facts and allocated fact in Power BI and in an YT video.

Discrepancy snapshot fact
A discrepancy snapshot fact table describes a state or a status of things at a particular time and when the state or the status does change there is a discrepancy. It is one way to capture versions of the fact over time with no need to keep all the daily snapshots from periodic snapshot fact.

A discrepancy snapshot fact table stores measures and things that provide context for the measures goes into dimension tables, therefore it is also possible to have a discrepancy snapshot dimension with discrete textual values and non-additive numeric values that can't be summarized, reminiscent of type 5 with a foreign key back to the main dimension. (Discrepancy snapshot in danish afvigelser).
Discrepancy snapshot fact example.

Data entry action snapshot fact – a way of Change Data Capture CDC
A data entry action snapshot fact table keeps track of a state or a status of things at a particular time as a logbook with three actions as: Added for a new entry, Changed for an existing entry and Removed for a non-existing entry that has been deleted or marked deleted in a source system and is not anymore a part of the full payload to the data warehouse. This snapshot approach has a dimension for data entry action with the three mentioned action values (could also have been: Influx, Changed, Departure).

The data entry action snapshot fact can be used to show the added/gain values from the latest ETL process or the removed/lost values and in combination with

changed values from one state to another state.
Formula: current = old current + added/influx – removed/departures
(In danish: igangværende = forrige igangværende + tilgang – afgang
for bestand, nytilkomne, indgået og udgået, ikraft og udtrådt).
Data entry action snapshot example.

Derived fact or Additional fact
A derived fact table is created by constraining, slicing, dicing, grouping, pivoting, calculating, summarizing or aggregating (data enrichment) an existing fact table or by merging existing fact tables to obtain an additional fact table with specific subject data.

Kimball recommended fact data at the lowest detail grain as possible to ensure maximum flexibility and extensibility. Kimball calls it a first-level data mart with base-level facts as a basis for a second-level data mart with an aggregate-level fact or a consolidated fact or a derived fact.

A derived fact table is created for performing an advanced mathematical calculation and complex transformations on a fact table like for a specific KPI (Key Performance Indicator), or an aggregate fact with a summation of measures to a higher grain like from date level to month level and from product level to brand level and using shrunken dimensions for Month and Category as a dimension lifted to a higher grain from the base-level dimensions as Date and Product.

It is fine to create a report-centric derived fact table from a base-level fact table or from multiple fact tables, and please remember the data lineage.

A derived fact table can be based on multiple base-level fact tables to get faster ad hoc query performance and simplify queries for analysts and for providing a data set to the Presentation interface area.

Derived facts can have different grains (granularity) to fit the purpose for KPIs in dashboards and reports to achieve a faster performance by avoid doing performing costly calculations »on the fly«.

Be aware of drill-across and fan trap and chasm trap and loop trap can occurs when two fact tables are joined together, because there can be a fact table with no relation to another fact table except that each one contains a foreign key for a shared dimension table. Read more about traps and the unified star schema and join by a bridge table because having many fact tables each surrounded by some dimension tables, a single bridge table sits in the middle surrounded by all dimensions and facts.

Aggregate fact, Aggregated fact, Aggregate-level fact or Summarized fact
An aggregate fact table or a summarized fact table is a derived fact table as a pre-calculated fact that store computed summarized measures at a higher grain level of one or more dimensions to reduce storage space and query time greatly and eliminate incorrect queries. An aggregate fact is placed in a second-level data mart where it derive data from a base-level fact in a first-level data mart, and measures in an aggregate fact are a computed summary of measures from a base-level fact.

Dimensions in an aggregate fact can be derived from base-level dimensions and is called shrunken dimensions because the values are rolled up to create less fact rows e.g. Date dimension becomes a Month dimension, Product dimension becomes a Category dimension and an Address dimension becomes a Region dimension, and therefore the measures can be summarized to less fact rows for better query performance.

Year-to-Date ytd fact where month February is a summing up or roll up of January and February and so forth. Last-year-this-year fact with calculation of index compared to last year as a new column and easy to show in a report.

Aggregate fact table is simple numeric roll up of atomic fact table data built solely to accelerate query performance. It is called **incremental aggregation** when a ETL process does a dynamically update of a table by applying only new or changed data without the need to empty the table and rebuild aggregates.

Consolidated fact
A consolidated fact table is a derived fact table that combine data from multiple business processes and their fact tables when they expressed at the same grain, and it used to combine fact data from multiple source systems and several business processes together into a single consolidated fact table. A consolidated fact is placed in a second-level data mart where it derive data from multiple base-level facts in first-level data marts. A ETL process for a consolidated fact is not easy to develop a data integration (merger) with a good performance but it give a BI tool a fast performance.

For example, sales actuals can be consolidated with sales forecasts in a single fact table to make the task of analyzing actuals versus forecasts simple and fast to compare how the year is going.

Read more about consolidated dimension in section 4.4. Types of dimension tables and about consolidated mart in section 1.12. Data mart category.

Consolidate is to combine into one whole. (In danish at konsolidere, integrere, slå sammen, sammenlægge, sammenstille, samle til et hele, fuldstændiggøre).

Exploded fact
A exploded fact table contents huge number of rows where a period e.g. from a »date of employment« to a »termination date« as columns in one row, is going be turned around with one row per day of the period, or per week or per month depends of the wanted grain of the period, it can give a lot of fact rows, when an employee has 10 year anniversary it will make more than 3650 rows on day grain. Employee dimension type 2 to keep history of different names, departments and job functions.

For a windturbine power production the grain would be 15 minute grain which gives 96 rows per day for a windturbine power production.

When a exploded fact is a source for a OLAP cube it can sometimes be implemented as a view in the database, and when it is for an ad hoc reporting it will be used several times per day then it must be a materialized view stored in a table or sometimes as a indexed view.

Smashed fact
A smashed fact table contents several measures but only one or few of them has a value in each fact row. For the fact rows with same dimension member repeated in multiple contiguous rows with identical values, they will be smashed or collapsed into one fact row using operation as sum, min or max to limit the number of rows in the fact.

Column wise fact and Row wise fact
A column wise pivoted fact table is useful to be columns in a report e.g.
Revenue Jan, Revenue Feb, Cost Jan, Cost Feb, Sales Jan, Sales Feb.
For a OLAP cube, a row wise is much better because it gives good dimensions, e.g.
Period, Entry, Amount where Entry contains: Cost, Sales, Revenue.
Therefore a data warehouse needs to convert from columns to rows or vice versa.

4. Dimension
A dimension provides context descriptive data to a fact with measurement.

A dimension table contains business element with context by descriptive data values, and a dimension table is referenced by multiple fact tables so the containing measurements make sense to business users in different business functions and departments in a company or an organization.

A dimension table is a flattened denormalized table with a primary key in a column as an auto-generated unique sequence number as a surrogate key, which is called a dimension key.

A dimension has a value or member in a column, and values or members in a row.

4.1. Purpose of a dimension
Some purposes as I seen it:
- Entry point for a business user to query and analysis data in a data mart.
- Slice operation to fetch a specific data value from a dimension, 2014 Q1.
- Dice operation to fetch specific values of multiple dimensions, 2014 Q1 and Q2 and cities London and Washington D.C. and a product type Smart phone.
- Slicing and dicing where the dimension values (members) are selected and become a filter, a zoom or a search criteria to do a filtering of the fact measure data and analysis variable after a wish from a business user (search filter criteria in danish filtrering, søgekriterie; slice skære i skiver; dice klemme ud eller se data som en terning eller som en del-kube).
- Dimensions are placed in the preposition and form rows and as columns and form a cross tabulation or pivot for presentation and reporting of analysis variable or measure value divided by dimension values.
- Drill-down for presentation of dimension values in gradually greater detailing through hierarchy with levels, group with groupings, band with intervals, e.g. from a quarter to months or from a country to cities.
- Drill-up or roll up views dimension value at higher aggregation level and is summing up the fact measure through hierarchy with levels, group with groupings, band with intervals, e.g. from days to months to quarters or from cities to municipalities to countries.
- Drill-through operation to fetch the detail level data from a overall level aggregate data or summary data to show the underlying data from a source

system e.g. document number (in danish bilagsnummer).
- Drill-across ties data marts and facts together through conformed dimension and different fact measures can be compiled.
- Ranking or Sorting presents dimension values according to an analysis variable, e.g. customer segments in descending order after the sale.
- Top or bottom for presentation, e.g. top 5 products or 10 poorest customers on sale.
- Contains relevant data columns for reporting, analysis and self-service BI.

Hierarchy with levels – Hierarchical structure – Flattened hierarchy
The dimension values can be stored in a hierarchy with levels in columns.
For example, a Location hierarchy with three levels Country→Region→City
or a Product dimension with a hierarchy as Category→Brand→Label
where the hierarchy maximum dimensionality is 3, level of dimensionality is 3.
A dimension normally contains one or several separate and independent hierarchies with different numbers of levels to fulfill business needs from the business users.

Group with groupings
The dimension values can be stored in a group with groupings in one column.
For example, a horoscope twelve signs are grouped into four elements, or some SalesDevices are grouped into some DeviceModels.
A dimension normally contains one or several separate and independent groups to fulfill business needs from the business users.

Band with intervals
The dimension values can be stored in band with intervals in two columns.
For example, person ages in custom buckets in a Age band with intervals of:
Child (0-9), Tween (10-12), Teeanage (13-19), Young adult (20-29), Adult (30-66) and Senior citizen (67-130).

Another example is a Product dimension with two hierarchies (here called group) and with a band to divide the unit prices:
- Product→Product group
- Product→Brand→Brand group
- Product→Unit price level

Data classification is the process of organizing data into categories, group or category is part of a categorization or grouping of data to make a dimension more user-friendly to see data of a dimension on an aggregated and summed level.

Of course a dimension can be non-hierarchy, non-group and non-band.

Different dimensionality covers the issue that not all combination of multiple dimension values are allowed in a fact and a data warehouse needs to make sure of the data quality.

Naming a dimension is an important task. For example, a dimension will contain a recipient of goods and a sender of goods, and these can be individuals, citizens, private persons, self-employed persons, natural persons, legal persons, companies, organizations or owners, etc. I find a dimension name such as RecipientSender to be too specialized, therefore I will ask business users for a more generalized name like Economic operator or Player (in danish aktør) to be a common name for the dimension.

4.2. Types of columns in a dimension table
There are several types of columns in a dimension table, and we start with some dimensional keys which will be in a dimension table.

Dimension key is a surrogate key for a primary key in a dimension table.

A dimension key is an artificial auto-generated unique sequence number or an identity column as a surrogate key for the primary key in a dimension table.

A dimension key column must not allow null, must have a constraint **not null**.

A dimension key column is represented as an integer or as a date.

A dimension key value is immutable (unchangeable).

It is the dimension's own responsibility to generate the next value in column dimension key. The value of a dimension key does not come from a key mapping table to remove dependence from a source system, or come from a key map table.

A dimension key from a dimension table is used as a dimension reference in a fact table where a dimension key column is a foreign key in a fact table. A fact table has several dimension key columns to several dimension tables.

Kimball prefers a suffix **_key** for a dimension key column in a dimension table as the primary key for a dimension. An alternative suffix as _dimkey or _dim_key.

A name of a dimension key is independent of the type of slowly changing dimension.

For example, a dimension key column in a Customer dimension table has column name **Customer_key** as the primary key of the table which is related to a Sales fact table with a dimension key column Customer_key column as a foreign key to the Customer dimension table.

Different naming
I have seen type 1 dimension with a dimension key labelled Entity key EK or Ekey to represent an entity in source data, e.g. EK_Customer or Customer_EKey.
I have seen a dimension key labelled as a row key named Rid for row identifier, e.g. CustomerRid, or labelled as a durable key named Did for durable identifier, e.g. CountryDid which is incorrect according to Kimball page 101: »The durable supernatural key is handled as a dimension attribute; it's not a replacement for the dimension table's surrogate primary key« which is called a dimension key regardless of dimension type 0 – 7.
Sometimes a dimension key is labelled as a surrogate key with a prefix for data warehouse DW_SK_ or SK_ or with a suffix _SK, _SKey, SK, SID, ID, UID unique identification, e.g. DW_SK_Customer or Customer_SK or CustomerSID.
I have seen type 2 dimension with a dimension key labelled History key HK or Hkey to represent a history reference over time, e.g. HK_Customer or Customer_Hkey.
Sometimes a dimension key is labelled as a historical surrogate key HSID.
For me »surrogate« is a characteristic or a property, and not a name for a column.

I prefer a suffix **_key** for a dimension key column in a dimension table.

A dimension key must not be data-bearing (in danish databærende), it must be meaningless, but for a date dimension and a time dimension I like to use a **smart valued dimension key**, e.g. a date 2013-12-31 represented as a date data type, or a value 20131231 represented as an integer data type, and a time 08:30 am as an integer value 830 and 08:30 pm as an integer value 2030. With these dimension keys I can avoid a join from a fact table to a dimension table to fetch and see a date and a time.

I have seen a dimension key store which generates a global number or RefId for each dimension key value across all the dimensions to achieve uniqueness. I am not fan of a most recent surrogate key map table from Kimball page 464 and 506.

I like to think a dimension table as an object that has its own methods or functions on data where each dimension object has its own responsibility to generate the next value for the dimension key _key. A durable key comes from a key mapping table.

Business key is from a source system where it can be a primary key or a secondary key with unique values.

For example, Social Security number (SSN), StatusCode and CustomerNumber.

A composite business key of multiple columns is still a business key.

A business key column(s) is represented as an integer or a string (text) or as an alphanumeric value, a boolean (bit), seldome a date, a datetime or a decimal.

A business key column must not allow null, must have a constraint **not null**.

I prefer a suffix **_bkey** for a business key column(s) in a dimension table.

Some data engineers like prefix such as BK_ for a business key column(s).

A business key value is either mutable (changeable) or immutable (unchangeable), and is meaningful for a human being where a business user prefers to use a business key value to identify an entity for a search criteria lookup value giving in a phone call to a company, a bank, a hospital or the government.

A business key has an embedded meaning and represent a unique object in the business. An alias is an Enterprise wide business key because the same value is used in multiple source systems. If a business key is immutable (unchangeable) in a source system then it is called a durable business key.

From a data warehouse point of view, a business key also represents a surrogate key from a source system where it often is a primary key as an artificial auto-generated unique sequence number or an identity column (id, uid unique identification) and is immutable and meaningless for a human being.

Together with a business key it is nice to have a business date to indicate a date of change of one or more data values connected to the business key.

A key mapping table can transform business key values either to an immutable surrogate business key e.g. a code which can be understood by a human, or to an artificial auto-generated unique sequence number or an identity column as a durable key which can be included in a dimension and in a bridge table for later use of current values.

Please notice, that a business key translation in a key mapping table to a durable key same as surrogate key does not give a key value to a dimension key in a dimension, because for having independence of the business key, and that a type 2 dimension dimension key has unique values for each row even though the same durable key value or business key value is standing in several rows.

Durable key is a surrogate key for a business key in a key mapping table and in a dimension table.

A durable key is a code which can be understood by a human, or is an artificial auto-generated unique sequence number or an identity column as a surrogate key for the business key in a dimension table.

A durable key column must not allow null, must have a constraint **not null**.

A durable key column is represented as an integer or as a code.

A durable key value is immutable (unchangeable).

I prefer a suffix **_dkey** for a durable key column in a dimension table.

A durable key will be used in a dimension to glue multiple rows together in a timeline for a type 2 dimension or a type 7 dimension. In the other types as an additional representation of a business key.

Descriptive data values become **dimension values** and **dimension data** as context description for textual, numeric, date, time, hierarchy, group, band to define and identify a fact, and is saved in columns in a dimension table and will be shown to a business user as descriptive data columns with values to explain fact data in a row from a fact table.

Audit and metadata columns for housekeeping and data lineage
Section 6.3 describes several columns: InsertTime, UpdateTime, InsertExecutionLogId, UpdateExecutionLogId, IsCurrent, IsDeleted, CorrectionTime, IsInferred, ValidFrom, ValidTo, RowChangeReason. ValidFrom and ValidFrom dates in a dimension table to represent a timeline period where a dimension value was current, read more in section 4.3. Section 4.5 describes columns IsInferred and CorrectionTime.

For data lineage a dimension will include audit columns from the Archive area as RecordSource, ArcRecordId and ArcGlobalId for traceability back to the archive.

4.3. Types of slowly changing dimensions

Source data is volatile data because they will change over time e.g. a customer change his name or address and a product change place in a hierarchical structure as a result of a reorganization. How can we support **evolving dimension** data when dimension values and instances normally will change over time because of the volatility in source systems?

The rate of changes can be divided into two kinds of classifications and afterwards we will be looking into techniques to handle and tracking changes and to capture its history and preserve the life cycle of source data in changing dimensions.

A dimension regardless of the type will have a **dimension key** that is an artificial auto-generated unique sequence number or an identity column as a surrogate key for the primary key in a dimension table to join on a fact table dimension key.

Please notice, that when a dimension table inserts a new row, the row will get a new value in the dimension key column, typically a higher number (+1) than the last row inserted.

Slowly Changing Dimensions SCD
Columns of a dimension that would undergo changes over time. It depends on the business requirements specification or the user story whether particular column history of changes should be preserved in a data warehouse/data mart.
(In danish stille og rolig ændrede (skiftende, foranderlige) dimensioner med langsom opbyggende historik).

Rapidly Changing Dimensions RCD

A dimension column that changes frequently over time. The Slowly Changing Dimensions technique can result in a huge inflation of the number of rows in a dimension table. One solution is to move a particular column into its own dimension table with its own dimension key in a fact table.
Fast changing dimensions or Quickly changing dimensions.
(In danish hurtig ændrede (skiftende, foranderlige) dimensioner med omfangsrig opbyggende historik).

The methodology to handle dimension values that is changing over time from the source systems is called **Ralph Kimball's eight types** of Slowly Changing Dimensions approaches and techniques.

Type 0

Keep original value and constant value, where a dimension value will never change, and when a value is changed in a source system, the original value is retained in a dimension. Type 0 does only to store the original value from a source system. Type 0 means that values are fixed, because once a row is inserted, it will never get changed or updated. It is called no versioning.

Please notice, that when a type 0 dimension table inserts a new row with a new value, it is not reflected in a fact table, because the historical fact table rows continue to reference the same dimension key value.

There is no overwrite of values in a type 0 dimension.

The type 0 approach is to store fixed values that will never change over time, but a fixed value can be corrected in case of an error in a source system, e.g. a correction of a mistyping, a typos or a misspelling of a name.

When we choose a dimension to be type 0, it is because we don't want to change the values. We want only original values for the lifetime of a data warehouse.

For example, a Yes/No dimension, a Time dimension, a Date dimension from a calendar, currency codes (ISO 4217) and geographic data with a hierarchy as Country→Region→City.
But can we be 100% sure of no change in dimension data, else we will use a type 1 dimension.

A value in a business key column will be in one row only.
There can be a constraint unique key (unique nonclustered index) for business key to ensure uniqueness and no duplicates.
There is a one-to-one relationship between business key and dimension key.

A key mapping table can transform business key values either to an immutable surrogate business key e.g. a code which can be understood by a human, or to an artificial auto-generated unique sequence number or an identity column as a durable key which can be included in a type 0 dimension.
There is a one-to-one relationship between durable key and dimension key.

First time a business key value enters a type 0 dimension, a ETL process will insert a new row with the business key value and the original descriptive data values.

Second time a business key value enters a type 0 dimension, a ETL process will ignore the business key value and the new current descriptive data values.

Inserting a new row into a fact table will carry out a lookup on a business key value in a dimension table to fetch a dimension key value for the new fact row.

A fact table row refers to a dimension key value to fetch the original values, as-began.

A view upon the dimension will provide the original values for the dimension key to join on a fact dimension key to fetch the original values, as-began.
E.g. a view name DimCustomer_Original.
(In danish oprindelige værdi).

This is the technique for Slowly Changing Dimension: »Retain original.«

Type 1

Keep current value, where a dimension value will change, and when a value is changed in a source system, the old current value will be overwritten in a dimension by the new current value in the same row in the dimension. Type 1 does not preserve historical descriptive data values in a dimension table. It is called no versioning.

Please notice, that when a type 1 dimension table updates a row with a new current value, it is not reflected in a fact table, because the historical fact table rows continue to reference the same dimension key value.

There is overwrite of values in a type 1 dimension.

Type 1 assumes that registered values, when data was happening, occurred, registered or took place in a source system or an operational system, will change over time in a normal data processing in a source system, therefore type 1 ensures a continuous improvement of registered values, which include a later filling in missing data from a source system

The type 1 approach is to store the active, actual, present, newest or latest of a value, most recent or current value from a source system. An old value will be forgotten. The history of an old value is lost forever. Historical descriptive data values are not preserved. This approach is suitable when historical changes are not significant or when tracking historical changes is not required.

A value can be corrected in case of an error in a source system, e.g. a correction of a mistyping, a typos or a misspelling of a name.

When we choose a dimension to be type 1, it is because we don't want to keep the old values. We want only current values for the lifetime of a data warehouse.

For example »types of employee skills« because old names of the types are not wanted in a report or a dashboard.

Another example of certain dimension values with an age as a smart valued dimension key with values from 0 to 130 years old, and a Age band with intervals of: Child (0-9), Tween (10-12), Teeanage (13-19), Young adult (20-29), Adult (30-66) and Senior citizen (67-130).
In case of a change to the Age band intervals we do not want to keep the old intervals, we will as a type 1 dimension just overwrite the Age band intervals with new intervals, e.g. Child (0-17), Young adult (18-29), Adult (30-49), Senior (50-70), Retired (71-130).

Another example is a Product dimension with hierarchies, and when a product changes name, product group, brand, brand group or unit price, the old values will be overwritten and forgotten. Unit price is often saved in a sales fact table where it is seldome overwritten.

Type 1 dimensions are normally for reference data as a rule of thumb.

A type 1 dimension wants to present values at any time as current data and »one truth«, therefore old values are not important.

A value in a business key column will be in one row only.
There can be a constraint unique key (unique nonclustered index) for business key to ensure uniqueness and no duplicates.
There is a one-to-one relationship between business key and dimension key.

A key mapping table can transform business key values either to an immutable surrogate business key e.g. a code which can be understood by a human, or to an artificial auto-generated unique sequence number or an identity column as a durable key which can be included in a type 1 dimension.
There is a one-to-one relationship between durable key and dimension key.

First time a business key value enters a type 1 dimension, a ETL process will insert a new row with the business key value and the current descriptive data values.

Second time a business key value enters a type 1 dimension, a ETL process will update the existing row by overwrite with the new current descriptive data values.

Inserting a new row into a fact table will carry out a lookup on a business key value in a dimension table to fetch a dimension key value for the new fact row.

A fact table row refers to a dimension key value to fetch the most recent, current values, as-is.

A view upon the dimension will provide the current values for the dimension key to join on a fact dimension key to fetch the most recent, current values, as-is.
The current view will show a **unique list** of most recent values which is handy for a dropdown box search criteria for a report and for a dashboard.
E.g. a view name DimCustomer_Current.
(In danish aktuelle, nuværende, gældende, seneste værdi).

This is the technique for Slowly Changing Dimension: »Current by overwrite.«

Type 2

Keep historical value and current value, all values as history, where a dimension value will never change, and when a value is changed in a source system, the old

current value will be remained in a dimension and the new current value will be inserted into a new row in the dimension. Type 2 preserves historical descriptive data values in a dimension table. It is called row versioning.

Nicholas Galemmo said: »The point of a type 2 dimension is not to keep dimension history, it is to represent facts using historical context. If there is no need to represent facts in a historical context, there is no need for dimensional history.« source.

Please notice, that when a type 2 dimension table inserts a new row with a new current value together with a new dimension key value, it is not reflected in a fact table, because the historical fact table rows continue to reference the old dimension key value.

There is no overwrite of values in a type 2 dimension.

Type 2 assumes that registered values, when data was happening, occurred, registered or took place in a source system or an operational system, will change over time in a normal data processing in a source system, therefore type 2 ensures a preservation of registered values. A type 2 dimension is a storage of historical descriptive data values and it is able to fetch registered values at any point-in-time.

The type 2 approach is to store the full history of a changed value from a source system. The history of all values are kept forever. Historical descriptive data values are preserved. A type 2 dimension must always be able to present the active, actual, present, newest or latest of a value, most recent or current value from a source system. This approach is suitable when historical changes are significant and when tracking historical changes is required to have a full historical tracking with no data loss.

A value can be corrected in case of an error in a source system, e.g. a correction of a mistyping, a typos or a misspelling of a name.

When we choose a dimension to be type 2, it is because we do want to keep the old values. We want registered values for the lifetime of a data warehouse, because we want to present data values as they were when fact data was happening, occurred, registered or took place in the operational systems and it is often with a transaction date or an event date else when data was entered into the fact.

For example, a Customer dimension with »addresses of customers' residences«, because old addresses and cities are wanted in a report or a dashboard back in time for statistic of where the customers lived at date of purchase.

Type 2 dimensions are normally for master data as a rule of thumb.

A type 2 dimension wants to present values at any time as current data back then and »current truth«, therefore old values are important.

A value in a business key column will be repeated in a new row every time a data value is changing.
There is a one-to-many relationship between business key and dimension key.

We achieve to have an unlimited history of values over time marked by timeline metadata columns as a pair of data type date or datetime that represents the span of time when a row with its values was the »current truth« in an active period with metadata columns:

- Effective date and Expiration date (Active date, Expired date, Expiry date)
- EffectiveFrom and EffectiveTo
- BeginDate and EndDate
- StartDate and StopDate
- ValidFrom and ValidTo (I prefer this pair of columns)

Timeline metadata columns ValidFrom and ValidTo store the effective dates of validity for a row in a dimension table and represents a period for the time when the value was current and active in a source system.

A dimension changes is normally tracked on a:

- **daily grain** with no more than one change per day,
  ValidFrom and ValidTo gets a date, e.g. 2015-05-08.
- **minute-second grain** where many changes could occur on a given day,
  ValidFrom and ValidTo gets a datetimestamp e.g. 2015-05-08 14:12:21.
- **weekly grain** or **monthly grain** when source data is loaded once a week or once a month.

First time a business key value enters a type 2 dimension, a ETL process will insert a new row with the business key value and the current descriptive data values and:

- with beginning of time to metadata column ValidFrom value 1900-01-01 and with end of time to metadata column ValidTo value 9999-12-31.

Second time a business key value enters a type 2 dimension, a ETL process will:

- update the existing row, the old current row, by setting metadata column ValidTo to a value from a date of change based on a business date value from a source system or an operational system, or on a current date value, and
- insert a new row with the business key value and the new current descriptive data values with same date of change value to metadata column ValidFrom and to metadata column ValidTo value 9999-12-31 to represent the new current row.

Kimball page 508 calls it no gap exists between rows of the same business key value, meaning that the old current row ValidTo column and the new current row ValidFrom column are using the same date of change value. Date of change and how to determine a new value in a ValidFrom column, read more in section 6.5.2.

Inserting a new row into a fact table will carry out a **date range lookup** on a business key value in a dimension table to be between timeline metadata columns ValidFrom and ValidTo to fetch a dimension key value for the new fact row with this criteria, where I refer to a fact table, but more often it is a staging table:

```
fact.Business_key = dim.Business_key AND
fact.TransactionDate >= dim.ValidFrom AND fact.TransactionDate < dim.ValidTo
```

A key mapping table can transform business key values either to an immutable surrogate business key e.g. a code which can be understood by a human, or to an artificial auto-generated unique sequence number or an identity column as a durable key which can be included in a type 2 dimension.
There is a one-to-many relationship between durable key and dimension key.

Inserting a new row into a fact table will carry out a date range lookup on a durable key value in a dimension table to be between timeline metadata columns ValidFrom and ValidTo to fetch a dimension key value for the new fact row with this criteria, where I refer to a fact table, but more often it is a staging table:

```
fact.Durable_key = dim.Durable_key AND
fact.TransactionDate >= dim.ValidFrom AND fact.TransactionDate < dim.ValidTo
```

Read about date range lookup in sections 4.5 and 6.2.2 and 6.5.3 type 2 dimension sql merge implementation using a staging table on the way to load a fact table.

There can be a constraint unique key (unique nonclustered index) for business key + ValidFrom or for durable key + ValidFrom covering ValidTo and dimension key to speed up a date range lookup.

A fact table row refers to a dimension key value to fetch the registered values of the dimension at the time when fact data was happening, occurred, registered or took place often by a date column in the fact table based on source data, or when fact data was entered into the fact table by a current load insert date, as-was.

A view upon the dimension will provide the registered values for the dimension key to join on a fact dimension key to fetch the registered values, as-was.
E.g. a view name DimCustomer_Registered.
(In danish registreret værdi).

A view upon the dimension will provide the current values for the dimension key to join on a fact dimension key to fetch the most recent, current values, as-is.
The current view will show a **non-unique list** of most recent values which is not handy for a dropdown box search criteria for a report and for a dashboard, because a value can be repeated for different dimension key values used in a fact table.
A current view is implemented with a self join on a business key or a durable key.
E.g. a view name DimCustomer_Current.
(In danish aktuelle, nuværende, gældende, seneste værdi).

I like to add a metadata column as a current row indicator or a current flag called IsCurrent with two values: 0 for historical and 1 for current to mark each row in a type 2 dimension.

Slowly changing means that dimension data changes slower than business events which will be a fact table, e.g. a fact table for exchange rates with countries' currencies with exchange rates in relation to Danish kroner DKK.

This is the technique for Slowly Changing Dimension: »Keep history in rows as full history by adding a new row.«

Type 3

Keep current value and previous value, where a dimension value will change, and when a value is changed in a source system, the old current value will be remained and will be stored in a Previous column (or Historical column) in a dimension, and hereafter the old current value will be overwritten in a dimension by the new current value in the same row in the dimension. Type 3 preserves limited historical descriptive data values in a dimension table. It is called column versioning as an extension to type 1.

Please notice, that when a type 3 dimension table updates a row with a new current value, it is not reflected in a fact table, because the historical fact table rows continue to reference the same dimension key value.

The type 3 approach is to store the latest value, most recent, current value from a source system along with a sufficiently limited amount of historical data in a previous value or few prior values.

A value in a business key column will be in one row only.
There can be a constraint unique key (unique nonclustered index) for business key to ensure uniqueness and no duplicates.
There is a one-to-one relationship between business key and dimension key.

First time a business key value enters a type 3 dimension, a ETL process will insert a new row with the business key value and the current descriptive data values.

Second time a business key value enters a type 3 dimension, a ETL process will update the existing row with the new current descriptive data values in other columns.

Inserting a new row into a fact table will carry out a lookup on a business key value in a dimension table to fetch a dimension key value for the new fact row.

A fact table row refers to a dimension key value to fetch the most recent, current values, as-is, and some previous dimension values, as-was.

A view upon the dimension will provide the current values for the dimension key to join on a fact dimension key to fetch the most recent, current values, as-is, and to fetch some previous dimension values, as-was.

This is the technique for Slowly Changing Dimension: »Keep history in columns as partial history by adding a new column.«

Type 4
When a group of columns in a dimension is going to change often as a Rapidly Changing Dimension, a type 4 will split a dimension to a Base-dimension that has slowly changed columns, and split to one or more separate Mini-dimensions that has often changed columns (volatile data values) to keep the fast changing values in its own table. Type 4 store all historical changes in separate historical data tables.

For example, a Citizen dimension with columns for Social Security Number (SSN), FirstName, MiddleName, Surname, Address and Residential area name will be a rapidly changing dimension or Kimball called it »rapidly changing monster dimensions« because many citizens change addresses every day and once a year the municipality replaces addresses in residential areas. A dimensional model could be a base dimension and several mini dimensions.

- Base type 1 dimension for Citizen with SSN, Name and Date of Birth where it okay to forget a former name.
- Mini type 1 dimension for Address with StreetName, HouseNumber, FloorNumber, ZipCode and City where it okay to forget a former address.
- Mini type 2 dimension for ResidentialArea with Name, ValidYear and other data for statistical use like hierarchy with levels or group with groupings.
- An incremental loading fact will include all three dimensions, and a helper table knows the current year residential area for an address to help to fetch a dimension key value from ResidentialArea.

Another example is a rapidly changing column such as a Personal_Income, you can avoid a lot of rows in a Mini-dimension for each time an income changes by implement a band with intervals, e.g. from $0 to $19,999, from $20,000 to $49,999, from $50,000 to $74,999 and above $80,000 giving five possible banded values.

The type 4 approach is to store separately the current value and the historical value to maintaining data integrity.

The Base-dimension can be either a type 1 or type 2 (most often type 1) and the Mini-dimension becomes a type 2. We get a »capture a relationship in the fact.«

A fact table row refers to a Base-dimension dimension key value to fetch the Base-dimension values.

A fact table row refers to a Mini-dimension dimension key value to fetch the Mini-dimension values.

Kimball Design Tip #53 describes a customer dimension.

This is the technique for Rapidly Changing Dimension: »Keep history in tables.«

Type 5
Builds on the type 4 where a type 2 Mini-dimension gets a view to fetch the current rows and the dimension key column is labelled current dimension key, and the Base-dimension is extended with the current dimension key column that refers to the view of current Mini-dimension rows.

Base-dimension and Mini-dimension can join without include a fact to save query performance. The join will be implemented in a view and used in a BI tool. We get a »capture a relationship in the dimensions« because the current dimension key becomes a type 1 column outrigger in a Base-dimension.

A Base-dimension row can access a current row in a Mini-dimension directly without joining through a fact table, therefore 4 + 1 = 5 type.

A ETL process must overwrite a current dimension key value in a Base-dimension whenever a current dimension key values in a Mini-dimension changes over time as a result of the type 2 Mini-dimension is adding a new row.

This is the technique for Rapidly Changing Dimension: »Outrigger current dimension.«

Type 6
Mixture of type 1 and type 2 columns therefore a good idea to suffix columns as _t1 and _t2 to know which columns can be overwritten in the current row.

Can also have column of type 3, therefore 3 + 2 + 1 = 6 type.

Type 6 act as type 2 of tracking changes by adding a new row for each new version but type 6 also overwrites _t1 columns on the previous row versions to reflect the current state of data by using the business key to join the new row with the previous rows.

This is the technique for Slowly Changing Dimension: »Hybrid.«

Type 7
All rows follow type 2 to keep track of history values with a dimension key column and an extra key column called a durable key follows type 1 for the current value.

There is no overwrite of values in a type 7 dimension.

A key mapping table can transform business key values either to an immutable surrogate business key e.g. a code which can be understood by a human, or to an artificial auto-generated unique sequence number or an identity column as a **durable key** which will be included in a type 7 dimension.

A value in a business key column will be repeated in a new row every time a data value is changing.
There is a one-to-many relationship between business key and dimension key.
There is a one-to-many relationship between durable key and dimension key.

A fact table row refers to a dimension key value to fetch the registered values of the dimension at the time when fact data was happening, occurred, registered or took place often by a date column in the fact table based on source data, or when fact data was entered into the fact table by a current load insert date, as-was.

Inserting a new row into a fact table will carry out a **date range lookup** on a business key value or a durable key value in a dimension table to be between timeline metadata columns ValidFrom and ValidTo to fetch a dimension key value and a durable key value for the new fact row.

A fact table row refers to a durable key value to fetch the most recent, current values, as-is.
A fact table row refers to a durable key value to fetch the original values, as-began.
A fact table row refers to a durable key value to fetch a historical values at any point-in-time, as-of, read more in section 6.3.

A view upon the dimension will provide the registered values for the dimension key to join on a fact dimension key to fetch the registered values, as-was.
E.g. a view name DimCustomer_Registered.

(In danish registreret værdi).

A view upon the dimension will provide the current values for the dimension key to join on a fact dimension key to fetch the most recent, current values, as-is.
The current view will show a **non-unique list** of most recent values which is not handy for a dropdown box search criteria for a report and for a dashboard, because a value can be repeated for different dimension key values used in a fact table.
A current view is implemented with a self join on a durable key.
E.g. a view name DimCustomer_Current.
(In danish aktuelle, nuværende, gældende, seneste værdi).

A view upon the dimension will provide the current values for the durable key to join on a fact durable key to fetch the most recent, current values, as-is.
The current view will show a **unique list** of most recent values which is very handy for a dropdown box search criteria for a report and for a dashboard, because the values will be distinct by the use of the durable key column in a dimension table of type 7.
E.g. a view name DimCustomer_Current_Unique.

I like to add a metadata column called IsCurrent with two values: 0 for historical and 1 for current to mark each row in a type 7 dimension.

This is the technique for Slowly Changing Dimension: »Dual Type 1 and Type 2 Dimensions and Dual foreign keys in fact for a given dimension.«

Correction of data

A dimension value may contain a mistyping, a typos or a misspelling

For example, a first name of a customer Kelli was wrong and will be updated to Kelly after few days in a source system. Is is Rian or Ryan or Bryan?

Another example is an incorrect address, e.g. Christmas Street, because it could have been either Circle, Court, Lane, Avenue or Boulevard.
(The Big Bang Theory apartment building is located at:
2311 North Los Robles Avenue, Pasadena, California).

Another example is a wrong date for a Date of Birth, Date of Death, Date of Issue, Date of Expiration, Date of Launching something.

For type 0 and type 1 it is fine to do a correction of data to achieve correct values in a dimension where a ETL process will update an existing row in a dimension table and overwrite dimension values and mark the row with a metadata column CorrectionTime.

For type 2 and type 7 a changed dimension value will per automatic mean an extra row in a dimension. I think it needs some consideration especially if a source system can inform a data warehouse about a correction of data, it is fine to do a correction of data to achieve correct values. Kimball page 479 and 516 says, it is okay to do a correction of data as make a type 1 overwrite changes in a type 2 dimension.

A staging table for a dimension can have a metadata column IsCorrected as boolean (bit) set to True (1) where a ETL process will not insert a new row in a dimension table, instead a ETL process will update an existing row in a dimension table and overwrite dimension values and mark the row with a metadata column CorrectionTime, and maybe tag the changed data with a reason in a metadata column RowChangeReason. (Kimball page 451 last line).

Sometimes a source system needs to make a redelivery of a data set which can lead to correction of data in several type 2 dimensions or type 7 dimensions.
(In danish berigtigelse er rettelse af en forkert oplysning, gøre oplysning korrekt).

RowChangeReason column can also be used to tell why an Employee type 2 dimension adds a new row, because an employee has changed office location and RowChangeReason gets the value »Relocation.«
(Kimball Design Tip #80 and Kimball Design Tip #90).

Views for Original, Current, Registered and Active values

It is recommended to have views around a dimension to provide values for:

- as-began Original values
- as-is Current values
- as-was Registered values
- as-only Active values

I have added as-only Active values because sometimes a business user ask me to show only values which is used in a fact table or several fact tables to limit the list

of dimension values in a dropdown box or listbox. For a type 2 dimension, this ends up with four views, e.g. view names:

DimCustomer_Current
DimCustomer_Current_Active

DimCustomer_Registered
DimCustomer_Registered_Active

An active view needs to involve one or several fact tables to do the filtering to the active values. Power BI does the filtering which means a report can take a moment to start up.

When a business user only wants the active dimension values, we can consider a metadata column IsNotActive in a dimension to handle that a row is not active in any fact tables.

Examples of type 0, type 1, type 2, type 3, type 4, type 6 and type 7

Type 0 for a Date dimension and a Time dimension and a Yes/No dimension because the values will never change.

Type 1 for a MaritalStatus dimension with values e.g.: Single, Married, Separated, Divorced, Widowed, Separated, Registered partnership and Abolition of registered partnership because in case a value will change, we don't want to keep the old value for a status.

Type 1 for a Car factory specification dimension in case some values are changed in a factory system due to a correction, the dimension wants to present these values at any time as one truth about data, therefore old values are not important.

Type 1 for a Customer dimension example is in section 6.2.1 where old values are not important.

Type 2 for a Customer dimension example is in section 6.2.2 where customer values as Name and Address from a date of purchase are important to keep for accounting, purchase statistics and analysis for marking. The example includes a date range lookup and a presenting of a non-unique list of customer current names.

Type 7 for a Customer dimension example is in section 6.2.3 has the same purpose as Type 2 to keep old values, and the example includes a date range lookup and a presenting of a unique list of customer current names.

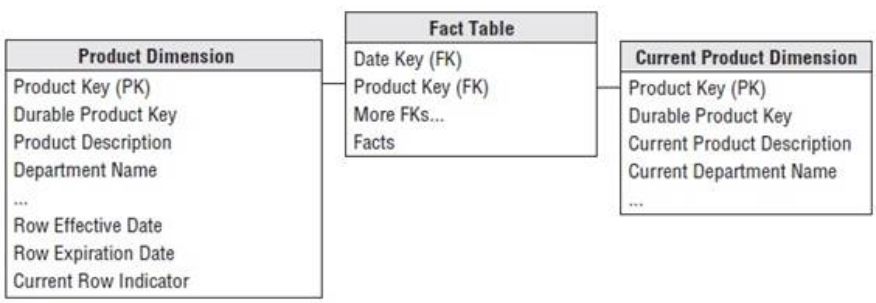
Type 3 for a Customer dimension example is three columns for postal codes with column names as Current Postal Code, Previous Postal Code and Oldest Postal Code. When a customer address is changed to a new postal code, a ETL process will move the value from Previous column to Oldest column and move the value from Current column to Previous column and add the new value to Current column.

Type 4 for a Customer dimension example where the customer values are split into a Base-dimension with slowly changes in columns CustomerName, ContactName, Address, City, Postal Code, Discount Band, Marketing category; and a Mini-dimension with fast/rapidly changes in columns Income, Raiting, AccountStatus, Days of payment.

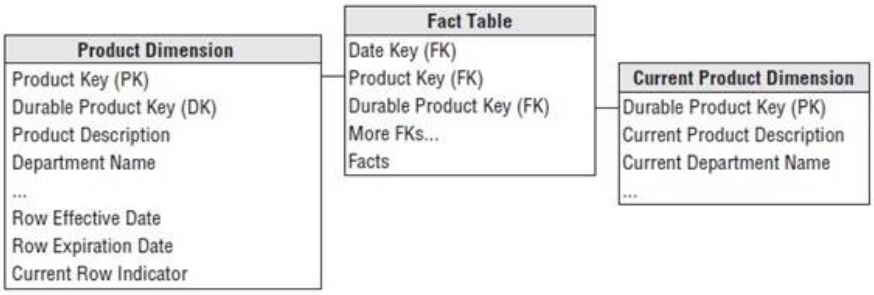
Type 6 for a Customer dimension example where the columns can be a mix of type 1 and type 2, so Name and Address is type 1 because we only need the recent value for shipment, and Postal Code and City is type 2 because we like to tracking these changes and keep values for the city a customer was living in when purchase a product. A »monthly sales in cities« report will use the city of the customers at the time a purchase was happening, occurred, registered or took place.

Diagram of type 2, type 4, type 5 and type 7

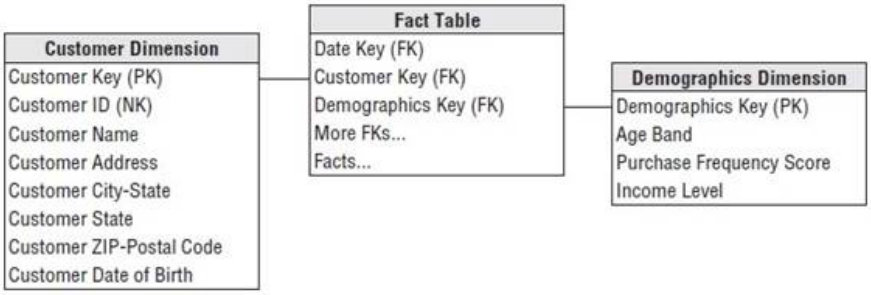
Type 2 diagram from Kimball page 163 with a dimension key in a fact table, and a current view will show a non-unique list of current value of products:



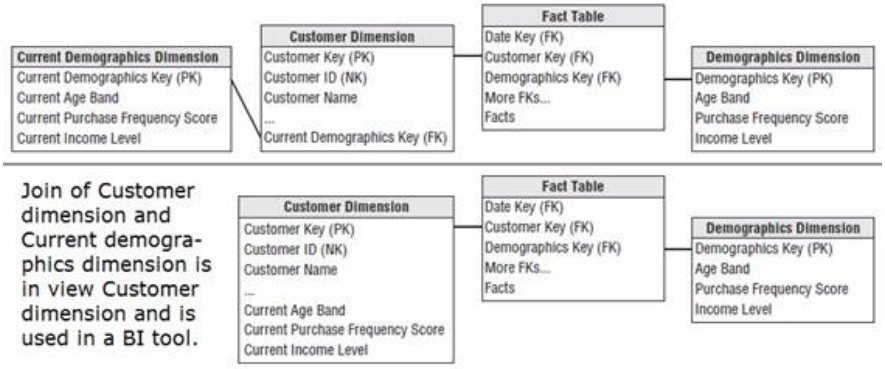
Type 7 diagram from Kimball page 162 with a dimension key and a durable key in a fact table, and a current view will show a unique list of current value of products:



Type 4 diagram from Kimball page 158 with Base-dimension Customer and Mini-dimension Demographics and both dimension keys are in a fact table:



Type 5 diagram from Kimball page 160 with Base-dimension Customer and Mini-dimension Demographics, and a current view will show a non-unique list of current value of demographics from the Mini-dimension with a current dimension key that is a type 1 outrigger column in the Base-dimension:



Choose a business key and a type of slowly changing dimension

With Bill Inmon's definition Non-volatile for a data warehouse or a data mart is that data do not change or historical data will never be altered, because it preserves historical data.

A type 1 dimension or type 3 dimension should be chosen with care, because they do not preserve historical descriptive data values in a dimension and it become volatile.

A type 2 dimension or type 7 dimension or the other types do preserve historical descriptive data values in a dimension and it become non-volatile. A dimension can always present the most recent, current values from a source system or an operational system.

A type 0 dimension can have all columns as part of the business key, for example a Person dimension does not have a natural business key such as PersonId, therefore Name, Address and City become a composite business key, e.g.:

Person_key	Name_bkey	Address_bkey	City_bkey	ValidFrom	ValidTo
140	John Watson	221B Baker Street	London	1881	1904
141	Sherlock Holmes	221B Baker Street	London	1881	1904
232	John Watson	14 Lodge Road	London	1905	1914
295	Mary Morstan	14 Lodge Road	London	1907	1912
306	Mary Morstan	14 Lodge Road	Leeds	1910	1923

Since all columns are part of the business key, it is a **type 0 dimension**, because there is no additional data columns outside the business key we can change. Since timeline metadata columns ValidFrom and ValidTo is given by a source system they make it look like a type 2 dimension and they can be corrected, but it is still a type 0 dimension, because there is no overwriting of values. One problem is that we don't know if there is one person, Mary Morstan, living in London and Leeds at the same time, or if they are two different persons. Person dimension becomes a »container of descriptive data values« instead of having the values as degenerate dimension columns in a fact.

It is a good idea to perform data cleansing with trim to remove space in a text string column and use a common formatting to avoid two rows like this:

Person_key	Name_bkey	Address_bkey	City_bkey	ValidFrom	ValidTo
73	Sheldon Cooper	2311 North Los Robles Avenue	Pasadena	2007	2019
76	Sheldon Cooper	2311 north los robles avenue	PASADENA	2007	2019

A simple method is to upper case Name and Address after trim, and use a hashbyte key value as the business key in the Person type 0 dimension.
Little advanced to say that »Avenue« and »Ave« and »AV« are same same.
Duplicate rows as shown above in a dimension makes it impractical for selecting a value in a dropdown box search criteria, e.g Pasadena and/or PASADENA.
After a data cleansing there will only be one row:

Person_key	Name_bkey	Address_bkey	City_bkey	ValidFrom	ValidTo
73	SHELDON COOPER	2311 NORTH LOS ROBLES AVENUE	PASADENA	2007	2019

I was missing the apartment number, because in October 2016 Sheldon moved in with Amy in Penny's apartment on the same floor with a different apartment number, therefore the type 0 dimension got an extra row. If we assume that the source system will send data about Sheldon's previous address and current address, we can fetch the existing row in the Person dimension to update the column ValidTo, and say the dimension becomes like a type 2 dimension without having the same business key value in two rows, for example:

Person_key	Name_bkey	Address_bkey	City_bkey	ValidFrom	ValidTo
73	SHELDON COOPER	2311 NORTH LOS ROBLES AVENUE, APARTMENT 4A	PASADENA	2007 Sep 24	2016 Oct 10
79	SHELDON COOPER	2311 NORTH LOS ROBLES AVENUE, APARTMENT 4B	PASADENA	2016 Oct 10	2019 May 16
74	LEONARD HOFSTADTER	2311 NORTH LOS ROBLES AVENUE, APARTMENT 4A	PASADENA	2007 Sep 24	2019 May 16

I've added Sheldon's roommate Leonard in apartment 4A to the example above.
In May 2018, Sheldon Cooper and Amy Farrah Fowler got married. What happens if they get divorced in February 2019 and Sheldon moves back in with Leonard?
The composite business key value already exists in the Person dimension, so a type 0 dimension would ignore the information. A type 1 dimension would overwrite the first row, and we would lose the data about where Sheldon lived back in 2007.
I think we need to include the column ValidFrom to the composite business key and say it is a type 0 dimension where column ValidTo can be updated, for example:

Person_key	Name_bkey	Address_bkey	City_bkey	ValidFrom_bkey	ValidTo
73	SHELDON COOPER	2311 NORTH LOS ROBLES AVENUE, APARTMENT 4A	PASADENA	2007 Sep 24	2016 Oct 10
79	SHELDON COOPER	2311 NORTH LOS ROBLES AVENUE, APARTMENT 4B	PASADENA	2016 Oct 10	2019 Feb 7
92	SHELDON COOPER	2311 NORTH LOS ROBLES AVENUE, APARTMENT 4A	PASADENA	2019 Feb 7	2019 May 16
74	LEONARD HOFSTADTER	2311 NORTH LOS ROBLES AVENUE, APARTMENT 4A	PASADENA	2007 Sep 24	2019 May 16
43	SHELDON COOPER	221B BAKER STREET	LONDON	1996 Jun 1	2049 Dec 31

Data profiling shows, that two people can have the same name, that a person can live in different places over a lifetime and that a person can move back to a previous location.

European customs authority uses an EORI Economic Operators Registration and Identification number as a unique identifier for each company for import and export of goods. An EORI-number is a perfect business key in a Company dimension of type 1 or type 2 or type 7. A company has a comprehensive guarantee with three

values cash deposit, undertaking given by a guarantor, another form of guarantee. A data modeler has added a comprehensive guarantee column to the type 2 Company dimension and included it in the business key, e.g.:

Company_key	EORI_bkey	Name	Comprehensive_guaratee_bkey	ValidFrom	ValidTo
1953	DK49562	Fast trade	Undisclosed	2022-01-01	9999-12-31
2107	DK49562	Fast trade	Cash deposit	2023-01-01	9999-12-31
2499	DK49562	Fast trade	Another form of guarantee	2024-01-01	2025-01-01
2824	DK49562	Ultra trade	Another form of guarantee	2025-01-01	9999-12-31

First problem is that the same company exists in multiple rows and a fact table row has no clue of the comprehensive guarantee and therefore lookup for Undisclosed with a datestamp to match timeline metadata columns ValidFrom and ValidTo.
Second problem is that the same company can have multiple names in same timespan. Since the grain of a dimension is determined by its business key, it is important to determine the right business key and the right type of slowly changing dimension. In this example I would have used a type 4 dimension with a Base type 2 dimension for Company with two rows for Fast trade and Ultra trade, and a Mini type 2 dimension for Comprehensive guarantee with three rows that can be referenced to from a fact table that knows about it. Company dimension will be extended with address, etc.

It is important to make a **data profiling** of a source system or an operating system to achieve an understanding of how data values are changing, and to know whether or not a data value will change or will never change over time in a source system, because it may choose a type for a dimension.

It is important to work together with the business users and understand their need for statistics and analysis in reports and dashboards to choose a type for a dimension, and talk about how basis data values should be handle in a data warehouse compared to the source systems or the operational systems.

Duplicate descriptive data values in a dimension

Product dimension is a type 1 with most recent, current values with a business key in column Number that is implemented as a constraint unique key. A mistake has give two products the same name marked in red.

Product_key	Number	Label	Brand	Category	UnitPrice
504	571145292	Blou jeans	Zara	Clothing	60
538	571145311	Comfie pants	Lee	Clothing	25
893	245600390	Feel awesome	Adidas	Shoes	89
1408	746153022	Feel awesome	Chanel	Beauty	37

Customer dimension is a type 2 to keep history in multiple rows when a customer changes name or city with a business key in column CustomerId_bkey that is implemented as a constraint unique key together with ValidFrom. Two different customers can have the same name marked in red.

Customer_key	CustomerId_bkey	CustomerName	City	ValidFrom	ValidTo
1	176	David Hall	London	1900-01-01	9999-12-31
2	359	David Hall	Boston	1900-01-01	9999-12-31
3	421	Marie Beaulieur	Lyon	1900-01-01	2007-11-16
4	421	Marie Beaulieur	Paris	2007-11-16	2014-07-15
5	421	Marie Sainte	Nice	2014-07-15	9999-12-31

Power BI has a slicer with distinct values, not show duplicate values, and it let you choose a product Feel awesome and a curent customer David Hall to see the total sales with behind the scene sql:

```
SELECT SUM(f.SaleAmount) AS TotalSaleAmount
FROM Fact.Sales f
    INNER JOIN Dim.Product p ON p.Product_key = f.Product_key
    INNER JOIN Dim.Customer_Current c ON c.Customer_key = f.Customer_key
WHERE p.Label = 'Feel awesome' AND c.CustomerName = 'David Hall'
```

What is the expected result? A mix of two products and customers sales, perhaps fair enough, but did the business user know that?

In a type 1 dimension as Product it is rare to have two products with same name, therefore I will make a unique constraint on Label column and not allow same name in two rows. A type 1 dimension lowest grain column is normally unique.

In a type 2 dimension as Customer it is fine to have two customers with same name because a person name is not unique, and some descriptive data values has changed, e.g. Marie Beaulieur moves from Lyon to Paris.

To distinct the two customers David Hall we can include their city in the slicer but it is not bulletproof when both are living in London, and a business user only wants to see total sales for one of them.

Slicer for current Customer is extended with column CustomerId business key:
David Hall - London - 176
David Hall - London - 359
Marie Sainte - Nice – 421

Behind the scene sql will use the unique business key:

```
SELECT SUM(f.SaleAmount) AS TotalSaleAmount
FROM Fact.Sales f
    INNER JOIN Dim.Product p ON p.Product_key = f.Product_key
    INNER JOIN Dim.Customer_Current c ON c.Customer_key = f.Customer_key
WHERE p.Label = 'Feel awesome' AND c.CustomerId_bkey = 359
```

I think you have been asked many times to provide a customer number to make yourself unique for an inquiry.

I wish a slicer could hold a dimension key or a durable key behind the scene and make a SQL:

```
SELECT SUM(f.SaleAmount) AS TotalSaleAmount
FROM Fact.Sales f
WHERE f.Product_key = 1408 AND f.Customer_dkey = 2 -- a type 7 dim in fact.
```

Please be aware of the possibility of duplicate descriptive data values in a dimension.

Alternative types of slowly changing dimensions

I have seen a type 2 dimension table with a dimension key called Rid for row identifier and a durable key called Did for durable identifier with a fact table only have the Did column, not the Rid column, to fetch the current values when IsCurrent = 1, therefore I call it a type 7~ (seven tilde). I guess it is inspired from Bridge table that use a durable key from a type 2 dimension, so the bridge does not change as dimension key change. Why not just have a type 1 dimension? I guess for another fact table it like to have the Rid column to represent facts using historical context.

I have seen a type 2 dimension split into two tables, where first table contains the current values and acts as a type 1 dimension, and second table contains all values, i.e. both current values and registered values and acts as a type 2 dimension. Each dimension table has its own dimension key and both of them are placed in a fact table. The first table represents a materialized view of a type 2 current view and will show a unique list of most recent values. I call it a type 4~ (four tilde). A ETL process will merge into both tables.

Some data modelers like suffix His or Hist for history as part of the name of a dimension or T1 and T2 for type 1 and type 2 dimensions, but I find that it is contrary to Kimballs naming where a dimension should generally appear independent of its type of dimension.

Kimball prefers a dimension table name to be independent of the types of slowly changing dimensions, therefore no indicator in prefix or suffix regardless of dimension type 0 – 7.

Temporal table system-versioned table

A temporal table with timeline metadata columns ValidFrom and ValidTo looks like a type 4 dimension because keeping history in a separate table and original table keeps current dimension members or values. But the dimension key is not an artificial auto-generated unique sequence number for each update of values in the dimension and therefore it can not be a type 2 to type 7.

Temporal table is only usable for a type 1, when it is including the deleted rows because older fact rows can be referring to it. We can say that a temporal table or a system-versioned table is a type 1 dimension with an addition of a history-tracking to handle historical data values for changes (updated or deleted) at any time. For example, I see no reason to have a type 2 Country dimension because it is very rare for a country to change its name, and when it does, most business users like to see older fact data with the current name. When you prefer to keep an older name for a country, you can to a type 1 Country dimension add the temporal table system-versioned table and use it for an ad hoc query together with timeline metadata columns ValidFrom and ValidTo to fetch one version of a country name.

Alternative to dimension key as auto-generated unique sequence number

A dimension key in a dimension table as the primary key of the table, is normally an artificial auto-generated unique sequence number or an identity column as a surrogate key to join on a fact dimension key.

An alternative to a sequence number is a hashbyte value for a dimension key, which is based on a compound key of columns Business key and ValidFrom, and it is translated to a unique hashbyte key value.

Inserting a new row into a fact table will use a normal date range lookup to fetch a dimension key hashbyte key value for the new fact row.

Hereby a dimension table can be truncated and reloaded without reloading the connected fact table, because the dimension key is a hashbyte value and ValidFrom column is determined by a Business date or by a Date of change in archive which is immutable (unchangeable).

Example of a composite primary key in a dimension in Kimball design tip #100

A Kimball perspective of Slowly Changing Dimensions

DistinctCount – Count distinct – be aware

Sales fact with dimension key columns to dimensions and columns with measures.

Purchase Date_key	Customer _key	Product _key	Quantity	SaleAmount	Customer _dkey
2005-10-12	1	504	2	84	1
2005-10-12	2	1408	1	27	2
2005-10-12	3	504	1	42	3
2010-04-24	1	893	2	178	1
2010-04-24	4	1408	2	74	3
2010-04-24	4	504	1	60	3
2015-08-16	2	893	3	267	2
2015-08-16	5	1408	1	37	3

I like to know the number of customers, I count over the customer dimension key from the Sales fact table and gets 5:
SELECT COUNT(DISTINCT Customer_key) AS NumberOfCustomers
DISTINCTCOUNT('Fact_Sales'[Customer_key])

When Customer dimension is a type 1 dimension it is okay to do a count.

When Customer dimension is a type 2 dimension it is not okay to do a count, because each value in column Customer_key in Sales facts does not represent a unique customer, e.g. value 3, 4, 5 refers to same customer Marie, because she has remarried and took her new husband's surname, but she has the same business key CustomerId_bkey value 421. I join to Customer dimension and I count over the customer business key and gets 3:
SELECT COUNT(DISTINCT CustomerId_bkey) AS NumberOfCustomers
DISTINCTCOUNT('Dim_Customer'[CustomerId_bkey])

When Customer dimension is a type 7 dimension it is okay to do a count, but I count over the customer durable key from the Sales fact table and gets 3:
SELECT COUNT(DISTINCT Customer_dkey) AS NumberOfCustomers
DISTINCTCOUNT('Fact_Sales'[Customer_dkey])

A Kimball type 7 dimension makes your life so much easier.

4.4. Types of dimension tables

There are several types of dimension tables.
A dimension can be characterized by one or more types.

Conformed dimension, Shared dimension, Common dimension, Master dim
A conformed dimension allows business users to consistently slice and dice metrics from multiple business process and fact tables with data from different source systems to be integrated based on common and unified columns.

A conformed dimension has the same meaning to every fact in multiple data marts, and measures will be categorized and described in the same way and ensuring consistent reporting across an enterprise data warehouse and is essential for enterprise data warehousing. Generalizing dimensions are important for an integrated data warehouse where a conformed dimension table is reused across fact tables.

It is desirable to achieve a dimension with conformed column names and conformed content and meaning, and shared across the entire enterprise data warehouse. Sometimes called a centralized dimension with a uniform definition for common data elements from multiple source systems with a standardized structure to get consistency in reporting by enables aggregation and comparison of metrics from multiple data sources to ensuring that reports do not suffer from discrepancies due to differing dimension definitions.

A conformed dimension is a consistent interface to make sure that data can be combined in a data warehouse and be used all over the business because values of a dimension means the same thing in each fact. With a conformed dimension we can combine and drill-across from fact to fact in one data mart or over several data marts, and analyze common columns and values. Separate fact tables can be used together with shared, common and conformed dimensions. Data integration creates conformed dimensions. A drill-across example for querying multiple fact tables and combining the results into a single data set in Kimball Design Tip #68.

Conforming of several source data is part of the integration to achieve a conformed dimension where data is integrated of different meanings and different columns must be compared against each other, rules must be set, and data must be cleansed to create a single version of the entity.

Conformed dimensions will unite and integrate data values among multiple source systems so it is easy to search across different types of data and sync them in a common report. Shared dimension is utilized in multiple fact tables in a data mart or across multiple data marts.

A key mapping table is useful to integrate business keys from multiple source systems to shape a conformed dimension with data across multiple source systems. Therefore I recommend to add a durable key into a dimension regardless of dimension type 0 – 7.

Dimension values comes from either a source system or is built by business rules in Usage supporting database. Non-conformed dimension can only be used within one fact. It is part of a ETL process to do conforming by merge, unite and consolidate multiple source data across the enterprise for making a conformed dimension.
[Kimball Design Tip #163 calls it a enterprise dimension.]

For example, a Customer dimension from different business areas for B2B and B2C with different business key values and with different addresses like a shipping address and a billing address. Sometimes data from a conformed dimension is send back to a source system as master data to be reusable in an organization.

Date dimension or Calendar dimension as a multi-level dimension
A very common dimension with the granularity of a single day with hierarchies as:

- Year→Half year→Quarter→Month→Date (five levels)
- Year→Week→Date (three levels) [a week not always belong to one month]

When a fact date is connected to the Date dimension, it is easy to make a report based on monthly level, quarterly level, yearly level or weekly level, or use a search filter criteria such as Q3 2013.

Date dimension date or day values can be stored in a group that roll up of periods into more summarized business-specific date group or date band.
For example Season or TimeOfYear (in danish Årstider):

- Winter (December January February)
- Spring (March April May)
- Summer (June July August)
- Fall (September October November)

Date dimension has normally many supporting columns like Weekday name, Month name and IsManagementReviewDay, IsCloseDay and IsHolidaySeason column has a data type of boolean (bit) with two values True (1) and False (0) as a flag indicator but to be used in a presentation tool it is much better to have a textual column, e.g. DayStatus with values »Holiday« and »Working day«. A rule of when a specific date is a holiday is depending on country, collective agreement, company policy, etc. is programmed into the stored procedure that builds a date dimension.
(Kimball Design Tip #119).

A fact table has often at least one column that represent a date e.g. an entry date, ordered date, purchase date, shipped date, delivered date, receipt date. When an event is happening it has normally a date or multiple dates like an injury with an InjuryDate, a ReceivedDate at insurance company, a RulingDate, a TreatmentDate, a BillingDate and a PaymentDate. **Date dimension is a role-playing dimension** and therefore the data mart contains multiple views upon the date dimension where each view can join to each date column in the fact table. Each view has a good name like the column in the fact table and the columns of the Date dimension is renamed to unique names for each view.

Dimension key in a date dimension can use an integer data type and values in a format yyyymmdd, e.g. 20131225 for Christmas day as a smart valued dimension key. In a fact table the date dimension key value is self-explanatory and is useful in a query for the month of December 2013 in sql: between 20131201 and 20131231.

Other integer values to handle these five **artifact values or inferred members**:

»Missing« (-1) when a source system provides a date value that is **null**, meaning it is not present in some data, not registered, not reported, »hasn't happened yet« or »be available later«, because the date to be determined is expected to be available later and fact table will be updated thereby. Therefore the fact table column for the date dimension gets the value -1 for »Missing date« as a not expected value. A missing date indicates that we would expect a date, which would not always be correct, see later under »Nothing«.
(In danish Mangler, Ikke angivet, Ubestemt, Uoplyst, Ikke forventet).

»Not available« (-2) when a source system provides a date value that is not known and not found in the data warehouse, because the date value does not exists in the data warehouse e.g. date value 1234-05-06. Therefore the fact table column for the date dimension gets the value -2 for »Unavailable date« or »Unknown date«.
(In danish Data værdien er ikke tilgængelig, Værdien eksisterer ikke og er ikke til rådighed i dimensionen, Ikke defineret, Ukendt).

»Not applicable« (-3) when the dimension is irrelevant for the fact row (N/A, N.A., NOAP or »Irrelevant«). Therefore the fact table column for the date dimension gets the value -3 for »Irrelevant date«.
(In danish Ikke anvendelig, Ikke relevant, Er ikke krævet, Ikke indberettet på detaljeniveau).

»Wrong«, »Incorrect«, »Bad«, »Corrupt«, »Dirty« (-4) when a source system provides wrong or bad date e.g. year of birth as 2099 or 2017-02-29 because year 2017 is not a leap year. Therefore the fact table column for the date dimension gets the value -4 for »Wrong date«.

»Nothing« (-5) when we know that it is fine that there is no date (null/empty) because a date is not required in a source system and therefore is not missing, not expected and therefore a date is not required in the fact table column. There can be other columns to let us know when a null/empty date represent missing or nothing. The date dimension gets the value -5 for »No date«. When the date dimension is role-playing for a BirthDate value -5 might stands for »Prefer not to say« or »No answer« or in a general way »No data«. Sometimes it is fine that there is no value (null/empty) because a value is not required in a date column, therefore I am using -5 for »Nothing«, and for another date column I am using -1 for »Missing«.
(In danish Ingen, Ingen tildelt, Ingenting, Intet, Ikke oplyst, Findes ikke, Blank, Tom, Uden data, Forventet ingen værdi fra tid til anden).

A smart valued dimension key in a date dimension will use a **date** data type and therefore a fact table with a date dimension will also use a date data type instead of the usual integer. When a fact table gets many millions of rows it can be a good idea to use **table partitioning** that is a behind technique to segment a fact table into smaller tables, and it is in most cases based on a dimension key from a date dimension where the dimension key will use a date data type to be able to create a partition range of months, e.g.:
'2010-01-01', '2010-02-01', '2010-03-01', … , '2050-11-01', '2050-12-01'.

Other date values to handle these five **artifact values or inferred members**:

»Missing« as 0001-01-01.
»Not available« as 0002-01-01.
»Not applicable« as 0003-01-01.
»Wrong« as 0004-01-01.
»Nothing« as 0005-01-01.
A future undetermined date is identified as 9999-12-31.

An artifact value or an inferred member in a dimension table is to ensure no null in foreign keys in a fact table by maintain referential integrity from the fact table to the dimension table.
Read about artifact values and inferred members in section 4.5.

(In danish artefact er et kunstprodukt, en genstand eller et fænomen der er skabt af mennesker ofte med et unaturligt eller kunstigt præg [kunstpause]. I softwareudvikling en fællesbetegnelse for ting, der er lavet i forbindelse med udvikling af software. Alternativt navn er markørværdi som markerer et mærke eller fænomen).

Example of dimension keys in a Sales fact with artifact values or inferred members

Ordered date key	»Not applicable« (0003-01-01, -3)
Purchase date key	2017-02-26
Shipped date key	1017-02-26 is »Not available« (0002-01-01, -2)
Delivered date key	2017-02-28
Receipt date key	2017-02-29 is »Wrong« (0004-01-01, -4)
Invoice date key	»Missing« (0001-01-01, -1)

Complaint date key	»Nothing« (0005-01-01, -5)

The customer did not make an order, because she purchased a product directly in a store at February 26th, 2017. She asked to have the product shipped home, but the date is a fake value from previous millennium, which needs to be investigated further. She has received the product, but there is a mistyping in the date that needs to be corrected. It is unexpected that an invoice is missing and has not been registered, and an accountant is looking at it. It's great that she hasn't had a complaint about the product.

Sometimes a date dimension uses the first day of a month to represent the whole month in a monthly budget fact table. Of course it is not an arbitrary date, but we must be careful of misleading, therefore never use an arbitrary product in a Product dimension.

Time dimension or Time-of-day dimension or Clock dimension
A very common dimension with hierarchies as:

- Hour→Minute→Second (three levels)
- Hour→Minute (two levels)

First hierarchy has the granularity of a single second with 86400 rows.
Second hierarchy has the granularity of a single minute with 1440 rows.

Time dimension values can be stored in a group that roll up of time periods into more summarized business-specific time group or time band.
For example Time division or Time interval (in danish døgn inddeling):

- Morning (6 am to 8 am)
- Rush hour (8 am to 11.30 am and 4 pm to 6 pm)
- Lunch hour (11.30 am to 1 pm)
- Afternoon hour (1 pm to 4 pm)
- Dinner hour (6 pm to 8 pm)
- Evening hour (8 pm to 11 pm)
- Night hour (11 pm to 6 am)

Time dimension is a role-playing dimension, e.g. Departure time and Arrival time.

Dimension key in a time dimension with granularity of a second can use an integer data type and values in a format hhmmss as a smart valued dimension key, e.g.:
00:00:00 is 0, 00:00:01 is 1, 00:01:00 is 100, 6:30:00 am is 63000, 8:25:30 pm is 202530 because 8 pm is also called 20 o'clock in the evening as 20:25:30, and 23:59:59 is 235959.

Another smart valued dimension key could be in seconds by the formula:
seconds = (hours × 3600) + (minutes × 60) + seconds.
00:00:00 is 0 second, 00:00:01 is 1 second, 00:01:00 is 60 seconds, 6:30:00 am is 23400 seconds and 8:25:30 pm or 20:25:30 is 73530 seconds, and 23:59:59 is 86399 seconds.

Dimension key in a time dimension with granularity of a minute can use an integer data type and values in a format hhmm as a smart valued dimension key, e.g.:
00:00 is 0, 00:01 is 1, 00:59 is 59, 01:10 is 110, 12:00 is 1200, 14:47 is 1447, and 23:59 is 2359.

Duration, e.g. 5:15 for minutes/seconds is sometimes written as 5.15 or as 5'15".

Other integer values to handle these five **artifact values or inferred members**:

- Missing is -1 »Missing time« when a source system provides no value as a null or an empty string.
- Not available is -2 »Unknown time« when a source system provides a time as 24:12:06.
- Not applicable is -3 »Irrelevant time« when a source system says so or a fact feels so.
- Wrong is -4 »Wrong time« when a source system provides a time as »late afternoon« or »when the sun goes down« or »the twilight hour« (in danish sen eftermiddag, når solen går på hæld, skumringstimen).
- Nothing is -5 »No time« when we know a time is not missing and therefore a time is not required for a fact row.

In a ETL process for making fact data rows, developed in a SSIS package, the column Time_key with data type smallint will be created as a derived column to the pipeline where null becomes -1 and time 10:34:45 becomes 1034, expression:
ISNULL(TransactionDatetime) ? (DT_I2)-1 : (DT_I2)DATEPART("hh",TransactionDatetime) * 100 + DATEPART("mi",TransactionDatetime).

Dimension key in a time dimension can use a time data type, but then there cannot be any artifact values or inferred members.

Yes/No dimension (True/False dimension, Boolean dimension or Flag dimension)
A very common dimension with the granularity of an answer with data values: »Yes« and »No«.

Yes/No dimension is a role-playing dimension, e.g. Signup, Patient follow-up plan, ConversationDueToAbsence.

Dimension key in a Yes/No dimension can use a boolean or a bit data type with two values True (1, Yes) and False (0, No).

[De facto standard that 0 represents False and 1 represents True as a boolean/bit. In SQL Server 7 we couldn't build an index on a bit column. In SQL Server 2000 we could in T-SQL build an index on a bit column but Enterprise Manager didn't give us the option, it first came in SQL Server 2005 Management Studio. Microsoft says, there is absolutely no point indexing bit columns because they can only have two values. For a large number of rows with 50% for 0 and 50% for 1, an index might buy you very little performance gain versus keeping the index up to date.]

Sometimes there is more than two answer options (1 and 0) based on other information, and a dimension key will use an integer data type and integer values to handle different answers, e.g. these **artifact values or inferred members**:

-4 for »No answer«
-3 for »Irrelevant answer« (»Not applicable answer«)
-2 for »Unknown answer«
-1 for »Missing answer«
2 for »Maybe«
3 for »Not sure«
4 for »I can't decide«
5 for »I don't know«
6 for »Prefer not to say«

Business industry classification dimension (Branche dimension)
A dimension with the granularity of a single segment with hierarchy as Sector→Line of business→Product group→Segment (four levels).
It is not easy to make a complete dimension with all values, therefore a level can have an artifact value or an inferred member as »Undistributed« to represent both a null and one or several unknown values. (In danish Ufordelt).

Regular dimension with a flattened structured hierarchy
All dimension members or values (or branches in a hierarchy) have the same number of levels which makes the dimension symmetrical or balanced such as the Date dimension. A regular dimension has a flattened denormalized structure.

Fixed-depth hierarchy: Country→Region→City where the three levels is in separate columns. Several columns in the dimension table is not at third normal form (3NF), therefore the dimension table contains redundant data or duplicate data by nature.

A Product dimension table can have columns like Product_key, Code_bkey, Label and other descriptive data columns, a Category because products are divided into categories and each category has a Target group. When a category belongs to multiple products, the target group will be repeated, meaning it does not satisfy third normal form (3NF), but it is okay for a regular dimension.

Example of a Product dimension with hierarchy Target group→Category→Product.

Product_key	Code_bkey	Label	Category	Target group
5831	E-973	EVGA GTX Titan	Graphic card	Gamer
5832	G-108	GeForce 1080	Graphic card	Gamer
5833	R-929	Radeon R9 290	Graphic card	Gamer
7092	B-009	Besteker Q9	Karaoke mic	Singer
7093	C-008	CHUWI K8	Karaoke mic	Singer
9250	U-670	Ultra boost	Running shoe	Athlete
9463	T-001	Trek Madone 9.9	Racing bicycle	Athlete

There is three levels in the dimension hierarchy and it is balanced meaning that each Product (Code/Label) has a Category and each Category has a Target group.

For the Product dimension the Code/Label is the lowest level of granularity, and there is two many-to-one relationships because many products roll up to a single category and many categories roll up to a single target group.

Ragged dimension with a recursive or unstructured hierarchy
A ragged dimension has leaf members (the last level of a hierarchy) that appears at different levels of the hierarchy and therefore contains branches with varying depths and number of levels which makes the dimension asymmetrical or unbalan-

ced. A ragged dimension is implement as a Parent-child structure or with a hierarchy bridge table.

Variable-depth hierarchy: Europe→Denmark→Copenhagen with three levels and North America→United States→California→Sacramento with four levels representing continent→country→state→capital.

To get a hierarchy structure as a fixed level balanced regular dimension with a fixed-depth hierarchy I can make a dummy level for a »not applicable« state of Denmark and get this: Europe→Denmark→N/A→Copenhagen.
Denmark and France is divided into regions, Philippines is divided into provinces and Germany has states/lands, therefore the name of level »State« could be changed to a broader word like »Division« which covers the different countries' areas.

Parent-child dimension
To model a flexible hierarchical structure where some dimension values have different levels of hierarchies called unbalanced hierarchy or variable-depth hierarchy. Every values in the dimension have a related parent (mother) value, except the top value. Some values in the dimension is a child like a leaf of a tree. A value is also called a node, a top node or a leaf node.

For example, an Employee dimension where the parent is the manager and the children is the employees under the manager, and some employees are both a manager and have another manager above, and the chain of command can be different from department to department.

Another example is an Organization dimension where some departments have sub-departments and some teams have sub-teams, but there are also teams that don't have sub-teams. This is the strongest side of Parent-child dimension to model.

Illustration of Parent-Child from a danish application

Convert a Parent-child dimension to a Ragged dimension example.

Consolidated dimension, Consolidation dimension, Consolidate dimension
A dimension provides an additional layer or level of information in a hierarchy. It is normal to store multiple alternate hierarchies in one dimension.

Kimball said: »Combine correlated dimensions into a single dimension«, for example from section 3.3 (capture a relationship in the fact) we could have consolidate Retailer and Chain into one dimension with a hierarchy as Chain→Retailer including history to handle a retailer changes chain or a retailer changes name or changes data in other columns.

It is always a consideration to have one dimension or to have several separate dimensions e.g. a many-to-many relationship between addresses for ship-to and bill-to or between name of sales rep and customer, it is best to handle as separate dimensions and keep both of them in a fact table row, read more in Kimball page 175-177.

An university have admittees, students, alumni, staff, faculty, administration, donor's and each type of person has it's own set of special person information that would not be relevant to other person types. Different dimensional modelings:

- Customer dimension for admittees, students and alumni. Employee dimension for staff, faculty and administration. Donor dimension.
- Person dimension with role-play dimensions as views for each type of person with a metadata column PersonType or group metadata columns as IsCustomer, IsEmployee, IsDonor.
- Super-sub person dimension with sub dimensions for each type of person.
- Dimensions for each type of person.

Special circumstances
One person may be an alumni and a student at the same time, for example, an alumni studying for master degree. One staff may be graduted from same university and is a staff and an alumni at the same time.

See later about a junk dimension which is to consolidate many »small dimensions« into a single dimension to reduce the number of dimensions to a fact and in a data mart.

Snowflake dimension that makes Snowflake schema or Starflake schema
A dimension table normalization is called snowflaking to get several normalized dimension tables and some of them are connected to a fact table.
A dimension with a hierarchy is snowflaked, where each level becomes a new dimension.
A dimension with several columns is snowflaked, where a group of columns become a new dimension.
When a dimension is divided into several dimensions, they can be individually connect to different facts.

Snowflake a dimension into multiple dimensions and make it look like a single dimension with a view to join the dimension tables together and select columns. Use a materialized view in case of performance issue and let a ETL process in a data mart make a single dimension to simplify query from a business user and a BI tool.

Examples under points a) to i)

a) Splitting a dimension hierarchy into two or more dimension tables is called »Splitting hierarchies and hierarchy levels into multiple dimensions« or »Snowflake a hierarchy«. For example, a Customer dimension with a hierarchy as Country→Region→City will be split into three dimension tables so only column City remains in the Customer dimension that is connected to a Sales fact, and two Snowflake dimensions for Region and for Country with one-to-many from Country to Region and one-to-many from Region to Customer. Region and Country can be reused in other dimensions for Supplier and Store, or in a Sales fact to quick divide in regions and countries (not snowflake). Of course depending of business requirements specification. Another example is a Product dimension that is splitted into a dimension for Product Category levels and into a dimension for Product Brand levels. The two dimension keys will be snowflake keys (foreign keys) in the Product dimension. Or the two dimension become independent dimensions with no snowflake keys in the Product dimension instead the three dimension keys will be added to a Sales fact because that is the way business users think of them. The fact will capture a relationship among data entities.

b) A Product dimension with a Category hierarchy is used in a Sales fact and Inventory fact. For forecasting (budgeting) the data is generated at Category level. Kimball will not do snowflaking of Product dimension instead he roll up to Category dimension as a strict subset of Product that a ETL process must take care of.

c) Splitting a dimension with columns from multiple source systems or when some columns will be business user type-in from an application to enrich data and to avoid redundant data. See an example in section 6.1.

d) Splitting columns of a dimension table into smaller dimension tables with one-to-many relationships so data values fulfill and comply with 3NF and BCNF to avoid redundant data and denormalized structure. Snowflaking is normalization to 3NF. For example, a Brand dimension has an owner of a brand and we make a Owner dimension whose dimension key (primary key) becomes a snowflake key (foreign key) in Brand dimension as a one-to-many relationship from Owner to Brand. The Owner dimension can be included in an Advertisment dimension and a Campaign dimension with a snowflake key.

e) Splitting dimensions and move common columns to a general dimension is called an Outrigger dimension. For example, an Address dimension with street name, house or apartment number, postal code, city, county, area, region, country, gps coordinates and so forth to be used in other dimensions e.g. a Customer dimension with a shipping address and a billing address, a Building dimension with location address and an Employee dimension with home address. The Address dimension contains one row per unique address. A Sales fact can have a ShippingAddress_key and a BillingAddress_key as role-playing dimensions. Or a factless fact table with Customer_key, ShippingAddress_key, BillingAddress_key and effective and expiration dates for reporting track addresses of a customer (Kimball page 235). See more later.

f) Splitting a dimension with many columns into a Base dimension with common core columns and into one or more Subdimensions, where a base and a sub dimension has a one-to-one relationship, and a base dimension has a snowflake key (foreign key) to only one of the subdimensions. It can be called a context-dependent snowflake.

g) A Shrunken dimension is not a snowflake dimension, see more later.

h) Splitting a dimension with rapidly changing columns is not snowflaking, see Mini.

i) Kimball's financial services example starts with an Account dimension including data of products and branches but he choose to remove these descriptive data columns to form independent dimensions of Product and Branch and not snowflake the Account instead add them to the fact because that's the way business users think of them too. The fact will capture a relationship among data entities.

Some data modelers do not like Snowflake dimension because of the query performance with more tables in a sql join. I don't use the terms snowflake schema and starflake schema because for me snowflake belongs to a dimension and sometimes it is great to have a denormalized structure and other times it is good to think of the above points.

Outrigger dimension or Reference dimension

When many columns belong logically together in a cluster or group it is fine to do a snowflaking to avoid a repeating large set of data and therefore making a dimension smaller and stable. Sometimes a canoe or a sailboat is using a rig to achieve balance because they are very narrow and a cluster of columns is placed in a Outrigger dimension or a reference dimension because it's dimension key (primary key) will be a snowflake key (foreign key) in the main dimension, and there is no reference found in any fact table.

For example, a Product dimension has a column called LaunchDate that represent the date when the product will be available for the customers, and from that date the product can appear in the Sales fact. The LaunchDate column is a snowflake key to a Date dimension that becomes Outrigger dimension because Date dimension has many columns about dates, weeks, months, years and maybe fiscal columns too.

Another example is demographic data for each country which is providing with 50 different columns. When we using outrigger dimensions we let each dimension has its own core columns. Dimensions with Address or Location is an outrigger to other dimensions e.g. Customer or Stock house. A slicing and dicing of data.

Another example is from a bank with two kind of customers for Person and Company with common data as name and address and with specific data where a Person has social security number, gender and marital status and a Company has VAT identification number, industry code and turnover amount. A Customer dimension handle the connections to the fact tables and becomes a hub, an anchor or a party. Every time the bank gets a new customer, it will be set up in Customer dimension and the dimension key value will be reused either in Person outrigger dimension or in Company outrigger dimension where both of them have a one-to-one relationship to the Customer dimension.

Shrunken dimension or Shrunken Rollup dimension
A Shrunken dimension is a subset of another dimension columns that apply to a higher level of summary of an aggregate fact and the shrunken dimension key will appear in the fact.

Examples under points a) to e)

a) The Month dimension is a shrunken dimension of the Date dimension. The Month dimension would be connected to a forecast fact table whose grain is at the monthly level, while Date dimension is connected to the realized fact table.

b) A base-level Sales fact has a grain per date and product and is connected to a Date dimension with columns of date, month, year and a Product dimension with columns of names of product, brand and category. The Sales fact is derived to an aggregate fact with a grain per month and category and is connected to a shrunken Month dimension with columns of month, year and is connected to a shrunken Category dimension with a name column. The aggregate fact has columns for Month_key and Category_key and a summary sale amount of the Sales fact where a ETL process is using the Product dimension to roll up a product to a category and match the category to the Category dimension. Therefore both dimensions Product and Category have a category name column that becomes redundant data in the dimensional modeling. Therefore shrunken is not snowflaking because a snowflake dimension is on 3NF.

c) Shrunken roll up dimensions are required when constructing aggregate fact table. When a Sales fact has a daily grain the number of rows can become very large over time, therefore an aggregate fact summarized to monthly level will have less rows. Since the aggregate fact don't need detailed customer information, the Customer dimension can make new shrunken dimensions for Region out of address, for Gender group and for Age band, and the summarized fact data become even less rows.

d) Sometimes business users want few dimension values e.g. Red, Yellow and Green and they want Pink to become Red and Orange to become Yellow and so forth and the rest of the colours gets a residual value called Other. I will make a ColourGroup shrunken dimension with values: Red, Yellow, Green and Other. I will make a mapping table that will translate the colours e.g. Red to Red, Pink to Red, Yellow to Yellow and Orange to Yellow and the rest of the colours to Other. In the loading to the fact table I will let the colours pass by the mapping table to fetch the ColourGroup dimension to the fact table to obtain good performance for various statistics for the business users.

e) Store data of an aggregate fact in a column of a dimension for easy searching e.g. a year-to-date SalesSegment in a customer dimension with descriptive data values as Excellent, Super, Good, Average, Less, Nothing. A ETL process must ensure the column is accurate and up-to-date with the Sales fact rows.

Kimball Design Tip #137 shows how to create data for a shrunken dimension.

Aggregate dimension
Dimensions that represent data at different levels of granularity to give higher performance. Can also refer to hierarchies inside a dimension with a higher grain.

Derived dimension
Like a Month dimension that is derived from a Calendar dimension or we can say that Calendar has been reduced to Month, Year and Week with the start date of the week together with a week number and which year the week belongs to. A derived dimension can also be created by aggregating two existing dimensions.

In a hospital we can from a Patient dimension and an Employee dimension derive a Person dimension. A person can over time be both an employee and a patient or at the same time when the employee becomes sick and will be hospitalized.
Fact data can derive dimension data and it is called a degenerate dimension.

Previously, I showed a Date dimension and a Time dimension and with a combination of them I can create a new dimension to handle date and hour to be used in Power BI Impact Bubble Chart e.g. from 2016-08-22 10:00 to 2016-08-26 22:00.
A fact table or a view can have a derived column like DateHourInterval:

```
DateHourInterval = FORMAT(TransactionDatetime,'dd-MM-yyyy HH:00','en-US')
```

A view can make the data rows for the derived dimension:

```
CREATE VIEW DimDateHourInterval AS
SELECT DateHourInterval = FORMAT(CAST([Date] AS datetime) +
        CAST([Time] AS datetime),'dd-MM-yyyy HH:00','en-US')
FROM DimDate CROSS JOIN DimTime
WHERE Minute = 0 AND [Date] BETWEEN '2010-01-01' AND '2029-12-31'
```

Mini-dimension or Historical dimension
For Rapidly Changing Dimensions for managing high frequency and low cardinality changes in a dimension of fast changing volatile columns they are placed in a mini-dimension or historical dimension with its own dimension key column which will be included in the fact table. This approach is called type 4. A dimension table will be split into two tables, one with type 1 columns and the other with type 2 columns.

An example is a customer with columns for Name, Gender, DateOfBirth, Address and Country→Region→City is placed in a Customer dimension (can be type 1 or type 2), and the fast changing columns BodyWeightAtPurchaseTime and MonthlyIncome interval, e.g. $ 0-10000, 10000-25000, 25000-50000, 50000-99999 is placed in a mini-dimension called CustomerBodyWeightIncome with its own dimension key and a foreign key back to the main dimension as a type 5. The Sales fact will have two columns to provide data for a customer, a dimension key for Customer dimension and a dimension key for CustomerBodyWeightIncome dimension. Sometimes it is necessary to have two or more mini-dimensions if the columns is changing rapidly at different times.

Bridge table
In dimensional modeling, a bridge table connects a fact table to a dimension table in order to bring the grain of the fact table down to the grain of the dimension table. Bridge table is used to resolve many-to-many relationship between a fact table and a dimension table. Bridge table is used to represent a many-to-many relationship between a fact and a dimension. (Kimball Design Tip #142).

A bridge table is to combine data that have:
- A many-to-many relationship between a fact and a dimension.
- A multi-valued column in a dimension or in a fact at the conceptual level, multivalued dimension, break it down at the logical level to a bridge table.
- Multiple alternates and versions of hierarchies in a dimension.
- Ragged hierarchies or variable depth hierarchical relationship.

Bridge relates fact and dimension, and take care of a multi-valued column.
Factless fact relates dimensions, when dimensions have a relationship according to a business process event.

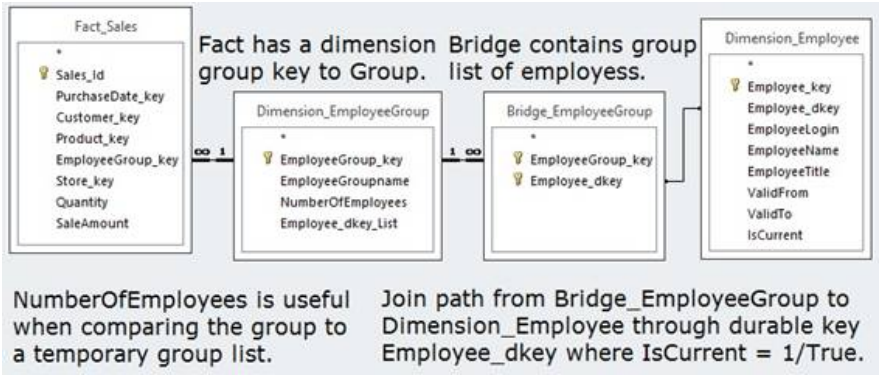
A bridge table is placed between a dimension table and a fact table when a single fact row can be associated to many dimension rows.

A bridge table connects a Group dimension table and a regular dimension table, and the Group dimension connects to a fact table. Bridge handles the multivalued.

A Group dimension is a type 0 dimension and it has fewer or more rows compared to the regular dimension.

For a regular type 1 dimension, the bridge table contains a dimension key to the dimension.

For a regular type 2 dimension, the bridge table contains a durable key to the dimension as shown in the Entity Relationship Diagram (ERD) below:



A row in Sales fact can have up to four different sales staff employees, and a sales staff employee has many sales. There is a many-to-many relationship between Sales fact and Employee type 2 dimension, where a sales staff employee becomes multivalued dimension in the Sales fact.

A row in EmployeeGroup dimension represents a unique combination of sales staff employees.

When you are looking at one row in Sales fact, you can take the dimension key value in column EmployeeGroup_key to fetch a EmployeeGroupname and the Number of employees in EmployeeGroup dimension. You can take the dimension key value in column EmployeeGroup_key to fetch the members of the group, sales staff employees, in EmployeeGroup bridge that gives you one or many durable key values in column Employee_dkey to fetch EmployeeName for each sales staff employee in Employee dimension where IsCurrent = 1. The values in the column Employee_dkey comes from a key mapping table where the values are immutable (unchangeable).

A ETL process will create a row in EmployeeGroup dimension for every unique combination of sales staff employees for a sale, and add the **group dimension key** EmployeeGroup_key in the Sales fact and in the EmployeeGroup bridge to save who was the sales staff employess doing the sale at the same time to a customer in a store for a product. EmployeeGroup is a type 0 dimension, because no rows will ever been updated.

EmployeeGroup dimension has a column called **Employee_dkey_List** that contains a list of Employee_dkey values from EmployeeGroup bridge to represent the sales staff employess doing the sale. A dimension group row has a list of durable key values from Employee dimension, e.g. 23,56,142 (from a string_agg) to be a kind of a business key value for each group row. Column Employee_dkey_List is useful when comparing a dimension group list to a temporary group list based on the source data, where a Sales fact row has several sales staff employees to become a list to do a lookup in EmployeeGroup dimension in column Employee_dkey_List as a business key for a Group dimension.

For example in a sql statement with incremental load based on column LatestTS:

```
INSERT INTO Dimension_EmployeeGroup(EmployeeGroupname, NumberOfEmployees, Employee_dkey_List, LatestTS)
SELECT
  EmployeeGroupname = 'List of sales staff employees',
  NumberOfEmployees = COUNT(*),
  Employee_dkey_List = STRING_AGG(d.Employee_dkey, ',') WITHIN GROUP (ORDER BY d.Employee_dkey ASC),
  LatestTS = MAX(a.LatestTS)
FROM Archive_Sales a
      INNER JOIN Dimension_Employee d ON d.EmployeeLogin = a.EmployeeLogin AND d.IsCurrent = 1
WHERE a.LatestTS > (SELECT ISNULL(MAX(LatestTS), '1900-01-01 00:00:00.000') FROM Dimension_EmployeeGroup)
GROUP BY a.DateOfPurchase, a.CustomerId, a.ProductNumber, a.StoreCode  -- the grain of the fact.

INSERT INTO Bridge_EmployeeGroup(EmployeeGroup_key, Employee_dkey, LatestTS)
SELECT
  EmployeeGroup_key,
  Employee_dkey = value,
  LatestTS
FROM Dimension_EmployeeGroup
      CROSS APPLY STRING_SPLIT(Employee_dkey_List, ',', 1)
WHERE LatestTS > (SELECT ISNULL(MAX(LatestTS), '1900-01-01 00:00:00.000') FROM Bridge_EmployeeGroup)
```

Kimball Design Tip #142 has a **Code_List** column instead of a dkey_List (type 2) or a key_List (type 1), but when a code has many letters or is composite of two columns, I find a list of key values much easier to work with, and there is nothing wrong with having both lists in a Group dimension.

A Group dimension's Bridge rows are fixed rows and are immutable (unchangeable).

More examples of Bridge tables and Group dimensions in next paragraph.

In a ETL process the loading order will be:
- Regular type 1 dimension with a dimension key.
- Regular type 2 dimension with a dimension key and a durable key (dkey).
- Group dimension as a type 0 dimension with a group dimension key, and a key list for a regular type 1 dimension or a dkey list for a regular type 2 dimension.
- Bridge table with a group dimension key, and a dimension key to a regular type 1 dimension or a durable key to a regular type 2 dimension.
- When a Group dimension has a Code_List column it can be updated using the rows from the Bridge table together with string aggregate.
- Fact table with a group dimension key.

Kimball uses the term Group, source:
- in a dimension name, e.g. EmployeeGroup dimension.
- in a bridge table name, e.g. EmployeeGroup bridge.
- in a group dimension key name, e.g. EmployeeGroup_key.

There will be a join path from a fact table row with a group dimension key to a group dimension and join to a bridge table and join to a dimension to fetch some descriptive data values which can be in several rows because of a many-to-many relationship between fact and dimension.

Illustration of a ETL process for a Group dimension and a Bridge table.

Instead of the term Bridge, a term Combine or Helper can be used. Peter Chen calls it an associative entity.
A song can have a Verse 1, a Chorus, a Verse 2, a Chorus, a Bridge and ends with the Chorus again.

Alternative implementations based on Kimball Design Tip #124 to eliminate the group dimension and the bridge table is:
1) Identify a primary sales staff employee for a Sales fact row and add a role-playing dimension key PrimarySalesStaffEmployee_key to the Sales fact.
2) Create a SalesStaff pivoted dimension including four role-playing dimension keys called Primary, Secondary, Tertiary and Quaternary for referring to Employee dimension and add a dimension key SalesStaff_key to the Sales fact.

Multivalued dimension, Many-valued dimension or Multi-valued column
When one fact row has two or more dimension values from same dimension.
When one dimension row has two or more dimension values from another dimension.
We must keep the grain of a fact and of a dimension, and we do not want a multivalued column in a fact or in a dimension, and we do not like a column for each value because there can be more than ten values.

Examples under points a) to g)

a) A Sales fact can have up to four different sales staff employees, and a sales staff employee has many sales. There is a many-to-many relationship between Sales fact and Employee dimension where sales staff employee becomes multivalued in the Sales fact. Let's start with a type 1 Employee dimension.

An implementation is to create a EmployeeGroup dimension with a group dimension key EmployeeGroup_key and a column called EmployeeGroupname that represents a group of sales staff employees. Groupname will contain values: »One employee«, »Two employees«, »Three employees«, »Four employees«.

One group has many employees, and one employee can be part of many groups. It is called a many-to-many relationship between EmployeeGroup dimension and Employee dimension, and it is implemented in a EmployeeGroup bridge table with two dimension keys: EmployeeGroup_key and Employee_key.

A Sales fact is adding a group dimension key column EmployeeGroup_key to tell that a fact row has a group of employees. The group dimension key EmployeeGroup_key tells that there is a join path from a EmployeeGroup dimension over a bridge table EmployeeGroup bridge to the Employee dimension to fetch one or more employees and their descriptive data values.

EmployeeGroup is a type 0 dimension, because it will never change a row, only insert a new row for a new group with a new value in EmployeeGroup_key. Likewise for EmployeeGroup bridge will only insert rows with Employee_key values to the new value of EmployeeGroup_key.

EmployeeGroup dimension and Employee dimension will contain **artifact values or inferred members** like -1 to represent »No sales staff employee« in a sale where

a Sales fact row will get -1 in group dimension key EmployeeGroup_key column.

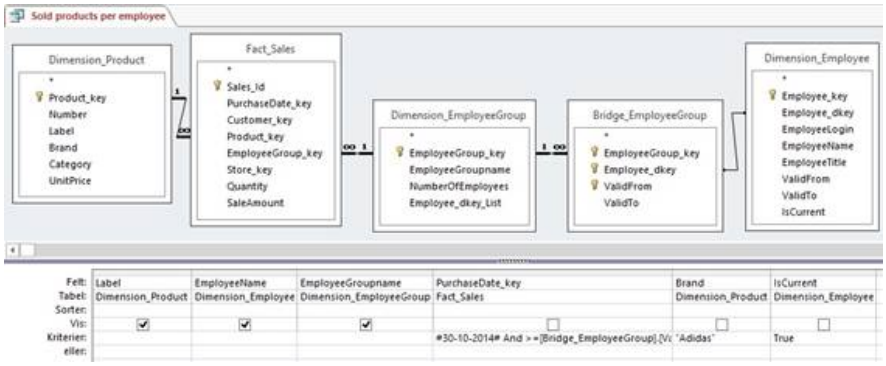
When Employee dimension is a type 2 and an employee change name, a new row will be inserted into the dimension with a new value in dimension key Employee_key. Nothing will happen to the rows in the EmployeeGroup bridge where dimension key Employee_key still refers to the old name of the employee and to fetch the current name of an employee then join to a DimEmployee_Current view.

When a fact row is added and it is referring to several employees from a type 2 dimension through a business key, I do not use date range lookup to fetch the dimension key Employee_key, instead I use the first value of Employee_key to avoid making a new row in EmployeeGroup dimension and new rows in EmployeeGroup-bridge with the latest value of Employee_key.

Nicholas Galemmo said: »In a bridge you would use the durable keys, so the bridge does not change as entities change.« source.

A durable key value is immutable (unchangeable) and comes from a key mapping table as a translation of a business key value.

An alternative implementation is to create EmployeeGroup bridge table with a group dimension key EmployeeGroup_key from EmployeeGroup dimension together with a **durable key** Employee_dkey from Employee dimension. The employee durable key becomes a join path compared to normal join on primary key and foreign key. Please remember, that a join from a Bridge on durable key to a type 2 dimension must be limit by metadata column criteria IsCurrent = 1/True.



An example of a sql statement to fetch employees who sell the Adidas brand.

```
SELECT p.Label, e.EmployeeName, g.EmployeeGroupname
FROM Fact_Sales f
   INNER JOIN Dimension_Product p
     ON p.Product_key = f.Product_key
   INNER JOIN Dimension_EmployeeGroup g
     ON g.EmployeeGroup_key = f.EmployeeGroup_key
   INNER JOIN Bridge_EmployeeGroup b
     ON b.EmployeeGroup_key = g.EmployeeGroup_key
   INNER JOIN Dimension_Employee e
     ON e.Employee_dkey = b.Employee_dkey AND e.IsCurrent = 1
WHERE f.PurchaseDate_key = '2014-10-30' AND p.Brand = 'Adidas' AND
    f.PurchaseDate_key >= b.ValidFrom AND f.PurchaseDate_key < b.ValidTo
```

Please notice, that the dimension key PurchaseDate_key is of the data type date.

It can be tempting to do another implementation by linking to another fact table, where we copy the primary key from the Sales fact which is a unique sequence number called Sales_Id and use it in a SalesEmployee factless fact together with Employee_key or Employee_dkey from the Employee dimension. But a fundimental foundation of a dimensional model is that a fact table is independent of any other fact table. Fact tables can be combined through common dimensions, but we don't design them with direct dependencies as we would with tables in a relational model. Each fact stands on its own, expansion to other subject areas is simple and each star schema is easy to understand.

Another implementation is to capture a relationship in the Sales Employee fact with dimension keys: PurchaseDate_key (type 1), Customer_dkey (type 2), Product_key (type 1) and Employee_dkey (type 2) to represent one or several employees for a sale of a product to a customer at a specific date. It is called a **factless fact** table.

Later on, I introduce a time-varying bridge which always must be considered when modeling a Bridge table, especially when a multivalued dimension is a type 2 dimension. EmployeeGroup bridge is including timeline metadata columns ValidFrom and ValidTo to have a time period or time span for when an employee was member of a group.

When data is deleted in a source system and therefore are gone forever, how is a data warehouse going to act?
· Incremental load to a bridge table and a Group dimension table because a deleting is very rare or source system does not inform about it.
· Full load to a bridge table and a Group dimension table for only current values.
· Use a time-varying bridge to have a timeline for registered values.

b) A business process: Buying a paint mixture the different colors are mixed with a ratio or weight (sum up to 100%) the amount of paint. A Sales fact row contains many colors and one color is included in many paint mixtures. Descriptive data values from a Color dimension becomes multivalued in a group of colors.

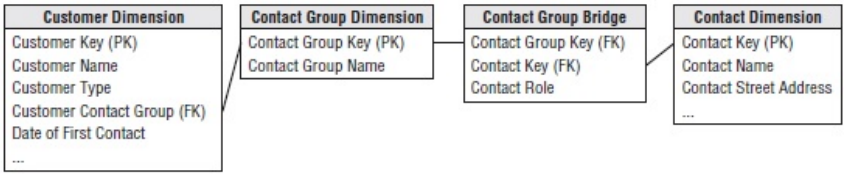
I create a SalesColorGroup dimension with group dimension key SalesColorGroup_key, and a SalesColorGroup bridge table with two dimension keys SalesColorGroup_key and Color_key and a Weight percentage rate to tell how much of a color is used in the group of colors. For a SalesColorGroup_key value the weight percentage rates rows will sum up to 100%.

The Sales fact is adding a group dimension key column SalesColorGroup_key to tell that a fact row has a group of colors. When a fact row like to fetch the colors, the group dimension key SalesColorGroup_key will join to SalesColorGroup dimension that will join to SalesColorGroup bridge table that will join to Color dimension to display the names of the colors that the fact row represents.

c) An employee can have several skills and one skill can belong to several employees. An Employee dimension does not like to have a multi-valued column for skills from a Skill dimension, which is called a multivalued subdimension in Kimball ETL Toolkit page 285. I create an EmployeeSkillGroup bridge table with a group dimension key EmployeeSkillGroup_key and a dimension key Skill_key, and Skill_key refers to Skill dimension. The Employee dimension is adding a group dimension key column EmployeeSkillGroup_key to tell that a dimension row has a group of skills.

When you are looking at one employee dimension row, you can take the value from EmployeeSkillGroup_key to fetch the members of the group, skills, in the EmployeeSkillGroup bridge table that gives you one or many Skill_key values to fetch a name of each skill in Skill dimension.

An alternative implementation is to have an EmployeeSkillGroup dimension with a group dimension key EmployeeSkillGroup_key that is adding to the Employee dimension to tell that an employee dimension row has a group of skills, and the EmployeeSkillGroup bridge and a Skill dimension. It is based on the example from Kimball page 249 that is including a Role column in the bridge table because a contact is associated with a specific role:



Star schema with examples of dimensions, bridge and fact with data values

An alternative implementation is that the Employee dimension does not have a column with skill, instead there will be created a **factless fact** EmployeeSkill with dimension keys Employee_key and Skill_key (assume both are type 1 dimensions).

d) In a Clothing mart a t-shirt can have up to three sizes »small«, »medium« and »large« and they can become three columns in a T-shirt dimension, or to create a Size dimension that has a many-to-many relationship to the T-shirt dimension and create a TshirtSizeFact with a measure QuantityInStock.

e) In a School mart I will place the students' courses in a fact table because it is a result of study and passed an exam can be noticed by a as a Graduate dimension.

f) In a Bank mart a bank account can be a joint account with two bank customers like wife and husband, and each bank customer can have several accounts like budget, saving and pension. There is a many-to-many relationship between a Customer dimension and an Account dimension, and I will create a BankAccount fact with dimension keys Customer_key and Account_key with measures for deposit, withdrawal and balance.

Let's say that both Customer dimension and Account dimension is a type 2, I prefer to capture a relationship in the fact with durable keys for Customer_dkey and Account_dkey, and a dimension key Yes/No_key role-playing dimension for Primary account holder, and timeline metadata columns ValidFrom and ValidTo.

An alternative implementation is based on a Customer multi-valued column in an Account dimension, because an account can have two or several bank customers. I create an AccountCustomerGroupBridge table with a group dimension key Account-CustomerGroup_key and a dimension key Customer_key. The Account dimension is adding a group dimension key column AccountCustomerGroup_key to tell that an account dimension row has a group of customers. Because Customer dimension is a type 2, I use durable key Customer_dkey in the AccountCustomerGroupBridge table.

A bank account can over time be a joint account, example a husbond adds his wife, or a mother gives bank power of attorney to her son, and later the arrangement can stop because of divorce or separation or death. AccountCustomerGroupBridge becomes time varying, a **time-varying bridge** by including timeline metadata columns ValidFrom and ValidTo to have a time period or time span to tell when a bridge table row was valid or was effective, e.g. when a customer has access to an account. Read more about how to handle a Customer type 2 dimension in Kimball Design Tip #136 and alternative implementation in Kimball Design Tip #166.

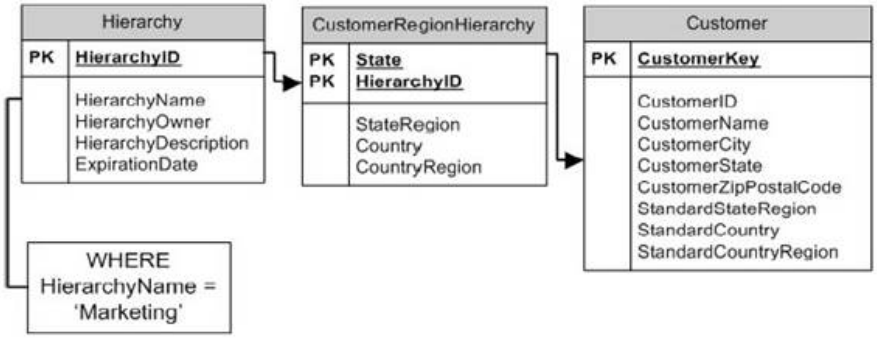
g) In a car insurance industry there is a CarDriver dimension with CarDriver_key and Name and Address for a cardriver and a driving licence number. A CarPolicy dimension with CarPolicy_key and many descriptive data values.

To handle several cardrivers in a family using the same carpolicy I create a Car-DriverGroup dimension with CarDriverGroup_key and I include CarPolicy_key. I create a CarDriverGroupBridge with CarDriverGroup_key, CarDriver_key and co-lumns for CarDriverType and Weight to contain values: »Single driver« 100%, for two drivers: »Main driver« 66,67% and »Second driver« 33,33%, and for three drivers: »Main driver« 50%, »Second driver« 25% and »Third driver« 25%, and BeginDate and EndDate for the time period a cardriver is included a carpolicy. The bridge table gives a picture of the drivers in a family.

An car insurance premium fact table with a grain of one carpolicy with measures for Period in months and PremiumAmount and dimension keys:
Date_key, CarPolicy_key and CarDriverGroup_key
to know the drivers and their weight of the premium amount.

Other examples in Kimball page 287 and 382 with a weighting factor.

Hierarchies in a dimension
A bridge table is used to resolve multiple alternates and versions of hierarchies in a dimension, called a hierarchy bridge table in Kimball Design Tip #62:



A hierarchy bridge table is used to resolve ragged hierarchies because it contains a row for each separate path from each node in the tree to itself and to every node below it. It is easy to for a parent node to fetch all its child notes and join them to a fact table roll up a business by parent node, make a roll up aggregation like a sum up. I know Kimball places a hierarchy bridge table between a dimension table and a fact table to illustrate a join path from dimension over hierarchy bridge to fact. Read more about »Help for Hierarchies« in The Kimball Group Reader from 2016 page 414-424, Kimball Design Tip #17, Kimball ETL Toolkit page 199-204, Kimball page 214-224 (some code). Hierarchy bridge table allows you to aggregate facts through a recursive hierarchy. An advantage is if a hierarchy changes, there is no need to re-associate because a hierarchy bridge table can be a time-varying bridge.

Role-playing dimension or Repeated dimension
A role-playing dimension is repeated two or several times in same fact e.g. a Date dimension which dimension key column is repeated in three dimension key columns in a fact table for three roles labelled SaleDate, ShipmentDate and DeliveryDate. For each role I will create a **view** with distinguishable, unambiguously and unique names of columns as renaming the columns from the Date dimension, and in this example it becomes three views upon the Date dimension called SaleDate,

ShipmentDate and DeliveryDate with columns like Date of Sale, Year of Sale, Date of Shipment, Year of Shipment, Date of Delivery and Year of Delivery.

A Flight fact has two dimensions Departure Airport and Arrival Airport, and they are role-playing dimensions to a Airport dimension with two views.

A City dimension can be repeated in multiple roles in a Person fact: BirthplaceCity, ChildhoodCity, ResidensCity, WorkingplaceCity, SeniorCity and DeathCity. It will become six views upon the City dimension.

A Manager dimension can be repeated as Sales clerk and Store manager.

A Sales fact can have columns as ShippingAddress_key and BillingAddress_key as role-playing dimensions.

In a fact table a dimension key column Date_key is a general term, therefore it is better to give it a prefix label and let it play a role, e.g. »purchase« becomes PurchaseDate_key in a fact table for a role-playing date dimension.

Another example is a Yes/No dimension with key values 1/True and 0/False and in different views for several role-playing statuses the two values are translated to good texts, for example: »Yes« and »No« or »In stock« and »Delivery made« or »Available« and »Utilized« for a Utilization status. An EmployeeAbsence fact can have a column as ConversationDueToAbsence_YesNo_key to indicate role-playing Yes/No dimension.

For a question like »Want children« the answer is normally »Yes« or »No« but we must give extra options as »Not sure«, »Maybe«, »I don't know« or »I can't decide« or »Prefer not to say« or »No answer« when the answer is nothing because it is okay that it is missing. Therefore we end up with a dimension with several values.

When a dimension has an outrigger dimension e.g. Customer dimension has a column for FirstPurchaseDate I create a view upon Date dimension and name it PurchaseDate. An Employee dimension with a HireDate I will create a view upon Date dimension and name it HireDate.

Kimball said: »Create the illusion of independent date dimensions by using views or aliases and uniquely label the columns.« It will be easy for a business user in Excel, Power BI, Tableau, QlikView, Qlik Sense, Alteryx or Targit to drag into a data model several dimensions without thinking of a dimension is playing multiple roles. In a OLAP cube data model the fact can be joined multiple times to the same dimension and at Dimension Usage each role can be labelled, but since we are using the same dimension the column names will be reused.

Junk dimension, Garbage dimension, Abstract or Hybrid dimension
A single table with a combination of individual and unrelated columns to avoid having a large number of dimension keys in the fact table and therefore have decreased the number of dimensions in the fact table. Kimball recommended up to 25 dimensions in a fact table.

A Junk dimension is used to fetch the descriptive data values for a fact table row and to avoid joining to multiple dimensions. A Junk dimension belongs to one fact only compared to a conformed dimension that is used in many facts.

A Junk dimension is a type 0 dimension with only values (names), or a type 1 dimension where a value (code) has an associated description (name) that can change over time and dimension only want to keep and display a current description.

Some data modelers like to suffix a name for a Junk dimension with Info, Profile or Property.

An illustration of a star schema with a Junk dimension named Order_Info to store code, flag and indicator columns to get a **lean Order fact** instead of having a big wide Order fact table.

There are two approaches for creating a Junk dimension based on Kimball Design Tip #48 and Kimball Design Tip #113.

The first approach is to create a Junk dimension table in advance with a content of the combination of all possible values of the individual columns.
It is called the cartesian product and in a sql called a cross-join.

For example, we have four individual, unrelated, miscellaneous columns with different business values, and we like to make the cartesian product for a Junk dimension:

Miscellaneous columns	Different business values

- Payment method Cash or Credit card
- Coupon used Yes or No or Not applicable
- Bag type Fabric or Paper or Plastic or Unspecified
- Customer feedback Good or Bad or None

For example in a sql statement to generate rows to a Junk dimension table:

```
;WITH
PaymentMethod(Payment_method) AS
 (SELECT 'Cash' UNION SELECT 'Credit card'),
CouponUsed(Coupon_used) AS
 (SELECT 'Yes' UNION SELECT 'No' UNION SELECT 'Not applicable'),
BagType(Bag_type) AS
 (SELECT 'Fabric' UNION SELECT 'Paper' UNION SELECT 'Not Plastic' UNION SELECT 'Unspecified'),
CustomerFeedback(Customer_feedback) AS
 (SELECT 'Good' UNION SELECT 'Bad' UNION SELECT 'None')
SELECT Payment_method, Coupon_used, Bag_type, Customer_feedback,
    ROW_NUMBER() OVER(ORDER BY (SELECT NULL)) AS Junk_key
FROM PaymentMethod
   CROSS JOIN CouponUsed
   CROSS JOIN BagType
   CROSS JOIN CustomerFeedback
ORDER BY Payment_method, Coupon_used, Bag_type, Customer_feedback
```

It will give 2 x 3 x 4 x 3 = 72 rows in a Junk dimension table and the rows will get unique number values from 1 to 72 in the junk dimension key column, illustration.

Remember to add some artifact values or inferred members, e.g. junk dimension key value -1 for missing when a fact row has a business key column with a null and therefore can't match a row in the Junk dimension.

A fact table needs only one dimension key to a Junk dimension to fetch the context descriptive data values for a fact table row.

The pitfall of a Junk dimension is the filtering, because a value (e.g. Credit card) exists as duplicate in multiple rows and therefore gives multiple dimension key values to be joined to a fact table.
To show unique content of a column from a Junk dimension in a dropdown or listbox we need to create a view for that column, for example:
Create view dim.PaymentMethod Select Distinct Payment_method From dim.Junk
The view will handle a one-to-many relationship to the Junk dimension.
A tabular model creates a calculated table upon the Junk dimension with a DAX:
Dim_PaymentMethod = Distinct(Dim_Junk[Payment_method]) and build a relationship from the calculated table Dim_Payment_method back to the Junk dimension in the tabular model where Junk dimension already has a one-to-many relationship back to the fact. Then hide the Dim_Junk because a business user does not need it after we have calculated tables for each of the columns in the Junk dimension, and therefore a Junk dimension becomes a bridge or a helper. A filter can do a distinct automatic.

For example in a sql statement to sum the fact rows with a Credit card:

```
SELECT SUM(f.SaleAmount) AS TotalSaleAmount
FROM fact.Sales f
   INNER JOIN dim.Junk d ON d.Junk_key = f.Junk_key
WHERE d.Payment_method = 'Credit card'
```

The second approach is to create a Junk dimension table with a content of the combination of all possible dimension values of the individual dimension keys in a fact table or several fact tables, or data is found in source data for a fact table.

Remember to add some artifact values or inferred members, e.g. junk dimension key value -1 for missing when a fact row has a business key column with a null and therefore can't match a row in the Junk dimension.

A Junk dimension of the second approach can have a derived column with enrichment data value based on a data rule, e.g. a compound value from columns from two dimensions which is the basis for the Junk dimension.

A Junk dimension can have a group with groupings in one column. If a Junk dimension has a hierarchy with levels in columns, I will prefer the columns to be in their own regular dimension with a fixed-depth hierarchy.

When a business user only wants the active dimension values, we can consider a metadata column IsNotActive in a Junk dimension to handle that a row is not active in any fact tables.

The extended approach is to create a Junk dimension table in a second-level data mart based on dimensions or base-level fact from a first-level data mart.

In a first-level dimensional model I prefer to have many dimensions to have 100% flexibility.

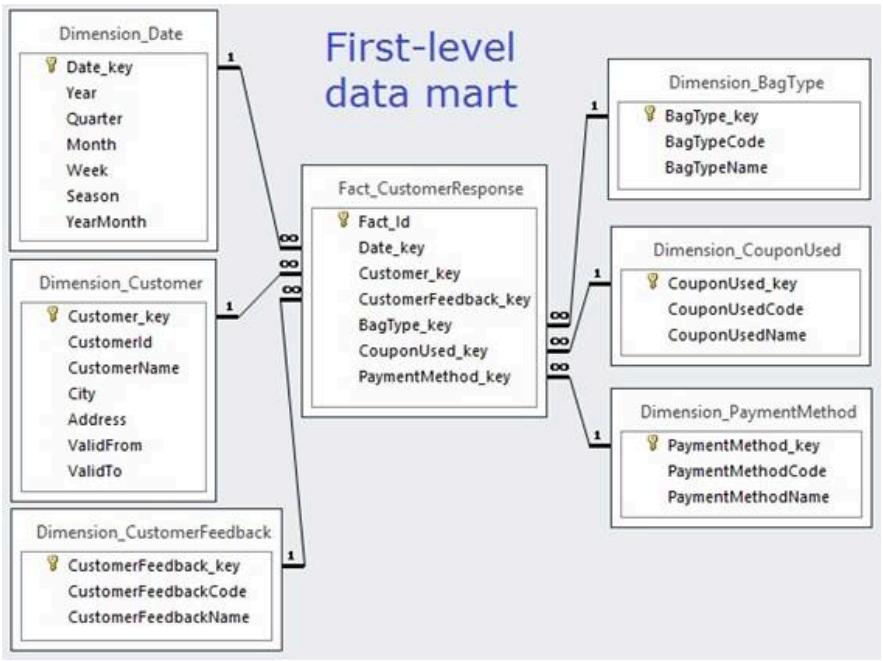
In a second-level dimensional model I prefer to have a Junk dimension based on dimensions from a first-level dimensional model where I have included their dimension key columns to be considered as a composite business key in the Junk dimension, and it gets its own junk dimension key.

The Junk dimension can be based of the first approach as a cartesian product of the existing dimension rows including their artifact values or inferred members. Dimension key columns and their values from the existing dimensions are included in the Junk dimension.

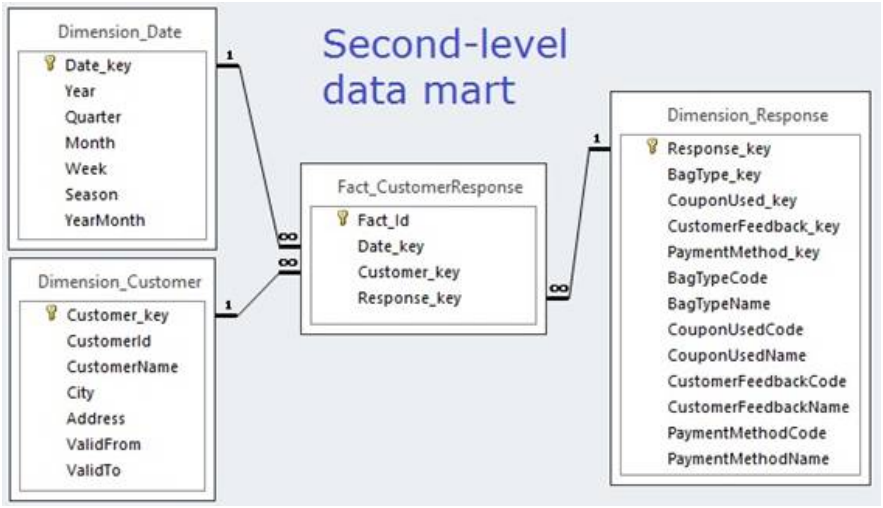
The Junk dimension can be based of the second approach, where a ETL process will fetch the dimension key columns and their values from a base-level fact by Select Distinct, and insert new combinations into the Junk dimension and gets the next number value in the junk dimension key column.

A ETL process for a second-level consolidated fact based on a first-level base-level fact with the relevant dimension key columns will do a lookup to the Junk dimension to fetch a junk dimension key value to a consolidated fact row.

An example of a first-level dimensional model with regular dimensions:



An example of a second-level dimensional model with a Junk dimension:



Degenerate dimension
Degenerate dimension values exist in a fact table in columns because there is no dimension table. When a textual or numeric value is stored in a column as part of a

fact table, it is typically used for lowest grain or high cardinality dimensions such as voucher number, transaction number, order number, invoice number or ticket number. There is no other columns, therefore a degenerate dimension is a dimension without descriptive data columns and no hierarchy, no group, no band.

A degenerate dimension is not a dimension table, but it can be implemented as a view with distinct values based on a fact table.

For example, an OrderNumber column in an Order fact table with several rows using the same order number value, because one order can contain several products. Therefore the OrderNumber column is important to group together all the products in one order. Later for searching for an order number in a OLAP cube, an Order number dimension is very useful, but it is not a dimension table, it is generated from the Order fact and there is no additional data like a name or text. OrderNumber can be used to trace back to transactions in the OLTP system.

Dimensionalised dimension or Degenerate dimension surrogate key
Kimball Design Tip #30 has guidelines to keep a fact table as lean as possible, and one of the advice is: »Take all text fields out of the fact table. Make them dimensions. Especially comment fields.« Avoid a big wide fact table.

Dimensionalised dimension is a replacement of Degenerate dimension, where a view becomes a materialized view meaning it becomes a dimension table in a data mart and a textual column will be transformed to an integer dimension key. Kimball page 101 calls it a Degenerate dimension surrogate key.

For example in a sql statement that insert rows to a new created table:

```
SELECT OrderNumber_key = CONVERT(BIGINT, HASHBYTES('SHA2_256', o.OrderNumber)),
    o.OrderNumber
INTO DimOrderNumber
FROM (SELECT DISTINCT OrderNumber
    FROM FactOrder
    WHERE OrderNumber IS NOT NULL) o

ALTER TABLE DimOrderNumber
ALTER COLUMN OrderNumber_key bigint NOT NULL

ALTER TABLE DimOrderNumber
ADD CONSTRAINT PK_DimOrderNumber PRIMARY KEY CLUSTERED (OrderNumber_key)
```

Dimension key OrderNumber_key will be added to a FactOrder table by the same convert code as above, and the textual column OrderNumber is dropped from fact.

The DimOrderNumber table gets added an inferred member with dimension key OrderNumber_key as -1 for missing to be assigned the FactOrder table when there is no OrderNumber text value. DimOrderNumber is a type 0 dimension.

Another example where I only have data for DescriptionOfGoods, and I do not like to add it into a fact table, therefore I transfer the textual column to an integer dimension key and insert it into a Goods dimension when the calculated dimension key value does not already exist.

For example in a sql statement implemented with Common Table Expression (CTE):

```
;WITH ds AS
(SELECT DISTINCT DescriptionOfGoods = TRIM(DescriptionOfGoods)
 FROM archive.Goods
 WHERE DescriptionOfGoods IS NOT NULL AND TRIM(DescriptionOfGoods) != ''
), gk AS
(SELECT Goods_key = CONVERT(BIGINT, HASHBYTES('SHA2_256', DescriptionOfGoods)),
    DescriptionOfGoods
 FROM ds
)
INSERT INTO dimension.Goods(Goods_key, DescriptionOfGoods, InsertTime_audit)
SELECT Goods_key, DescriptionOfGoods, sysdatetime()
FROM gk
WHERE NOT EXISTS(SELECT 1 FROM dimension.Goods d WHERE d.Goods_key = gk.Goods_key)
```

Dimension key Goods_key will be added to a FactOrder table by the same convert code as above, and the textual column DescriptionOfGoods is dropped from fact.

The Goods dimension table gets added an inferred member with dimension key Goods_key as -1 for »No description provided« to be assigned the FactOrder table when there is no DescriptionOfGoods text value. Goods is a type 0 dimension.

Static dimension or Constant dimension
Static dimensions are not extracted from a source system, but are created within the context of the data warehouse or data mart. A static dimension can be loaded manually with Status codes or it can be generated by a procedure such as a Date dimension and Time dimension. The opposite would be called Dynamic dimension.

Heterogeneous dimension
Several different kinds of entry with different columns for each fact (like sub classes). For example, heterogeneous products have separate unique columns and it is therefore not possible to make a single product table to handle these heterogeneous products.

Measurement type dimension or Entry type dimension or Fact dimension
Used to identify different facts that is populated in the same measure column in a fact table because the fact rows represent different entry typies. An measurement type dimension describes what the fact row represents and how measures must be understand and used. The alternative is to have multiple measure columns for each entry type-in the fact table where only one column has a value for each row.

Time machine dimension
A combination of two dimensions for Entry date (or event date) and for Year and where fact data rows is based on either »counterpart« as in a financial accounting or »transactions« as in a financial transactions. Section 6.4 will show an example.

Monster dimension
A very large dimension that has a huge number of rows or many columns. For a real estate agent, I implemented a 132-columns dimension through a merge of five source systems. The column names was made together with the business users. The created dimension table got the column names in an alphabetical order so it is easy to find a specific column.

Supernova dimension
Dimensional columns are allowed to become complex objects rather than simple text like unstructured text, gps tracking, graphic images and in time series and in NoSQL bases like Hadoop and MongoDB, and become much more malleable and extensible from one analysis to another. Extensible design in software engineering is to accept that not everything can be designed in advance and extensibility is a software design principle defined as a system's ability to have new functionality extended.

Audit dimension or Data quality dimension
A description of each fact table row would be »Normal value«, »Out-of-bounds value«, »Unlikely value«, »Verified value«, »Unverified value« and »Uncertain value«.

Kimball recommended to load all rows to the fact table and use an Audit dimension to do a tagging of data because of an error condition and thereby to tell the state of each row in an audit report so a business user can look at the data and make a data change or a counterpart in a source system and do a new load to the data warehouse to fix the data in the fact table. Data quality is monitored during a ETL process and it can procedure an audit statistics.

In a SSIS package a derived column can have an expression to validate data and give value to an Audit dimension where a sale amount from a source system is less than 0 or more than 5000 gets audit key value 12 for out of bounds amount else value 0 for okay with this formula:
SaleAmount < 0 || SaleAmount > 5000 ? (DT_I4)12 : (DT_I4)0
All rows are checked for compliance with the constraints.

An Audit dimension can also have data for name and version of a source system, name of data table in a source system, time of extract from a source system, time of insert into the fact table. (Kimball Design Tip #26, Kimball ETL Toolkit page 128, Kimball page 66, 192, 460, 511).

Numerical value as a fact measurement or as a dimension descriptive data
It is a matter of confusion whether a numerical value should belong to a dimension or a fact. For example, an unit cost price and an unit sales price of a product could be columns in a Product dimension, but since unit prices often varies over time and maybe also over location, it should be a non-additive measure column in a fact together with a Date dimension. (Kimball Design Tip #97).

Another example, a national lottery has a fixed pool size amount for each weekly drawing and since the amount is fixed and thus does not change over time, the pool size amount belongs as a column of the draw dimension together with a DrawDate column that becomes a outrigger or reference dimension which provide a week number and a year. A pool size amount can be used for filtering (like to see pool size above $1 billion) and for band intervals. When a person buys a lottery coupon, the deposit and the date of purchase will be recorded in a fact, and all the deposit amounts for a week can be summarized to a weekly deposit kpi, e.g. in a view or in a tabular model. A Power BI chart can combine the fact weekly deposit with the dimension weekly pool size and it can be seen that the deposit is significantly larger in the weeks that have very big pools.

4.5. Artifact values or Inferred members or Inferred dimensions

A dimension can have two inferred members (or artifact values) called »Missing« and »Unknown« to handle source data that is going to be loaded into a fact table:

- A fact business key has no value, it is null or it is blank or it is empty or it is nothing (•) or it is none or it is an empty string "". Null is valid to non-value.
  Approach: Handling a null/empty business key as a **missing** member because a row in a fact table has a column of a business key that is not registered or recorded. When it is fine that there is no value because a value is not required, I am using **nothing** or »no data«.
  For a yes/no question the answer is »Yes« or »No« (true or false, 1 or 0) where null represents »No answer« like the question was unanswered. A »Don't know« response is not the same as a neutral response therefore I treat the answer as a missing value, and I wouldn't keep it as a Likert scale item. »Prefer not to say« is neutral response.
  To treat null, blank, empty, nothing and empty string as equal to null in sql:
  WHERE (NULLIF(TRIM(CustomerAddress),'') IS NULL)

- A fact business key value was not found as a member value of a dimension.
  Approach: »Early arriving fact« for handling of orphaned data where fact data has an **unknown** member, meaning a fact value is an orphan child because there is no parent value in a corresponding dimension table, a fact value does not exists in a dimension because the value is unavailable or not available. »404 Document Not Found« means that a homepage deep link is referring to a page that does not exists or is unavailable because the page has been deleted or the page has been renamed or there is a misspelling in the link, therefore not available. It is known it is unknown.

In a relational database it is called a referential integrity constraint violation in a table when a foreign key contains a value that does not exists as a primary key in a different (or same) table. There is a no referred member and there is a need to have an inferred member. A forthcoming fact row has a member that will infer a new dimension member, therefore it is called inferred member of the dimension (in danish udledt), and it is to improve data quality and audit trail/control track for reconciliation in a data warehouse.

For example, an Order table has a foreign key column called PartNumber referring to a Part table, in case PartNumber column contains null it means you have an order without a part or the part is missing which for an order makes no sense. This is a mandatory relationship; you cannot order a part that does not exist, therefore an order's PartNumber value must exists in the Part table. In case you like to delete a part in the Part table its PartNumber must not being used in the Order table.

Handling a null/empty business key as a Missing member
When a data set is transformed to a fact table, there can be a business key column which value is empty or null, meaning it does not yet exist for the fact data. To keep all the fact data rows in the fact table, the related dimension table is already born with a »Missing« value (an inferred member) with dimension key value -1, which can be used as a default value in the fact table dimension key (foreign key) column. Later the fact table with a dimension key value -1 can be updated via a real business key value that either exists in dimension table or will be inserted first and get a new dimension key value that the fact data row can refer to.

Sometimes I have seen a dimension with an inferred member value in dimension key of -1 for »Unknown«, but I prefer using -1 for »Missing« (someone calls it »Blank«), and I am using -2 for »Unknown« to handle the situation of early arriving fact in general way. Sometimes it is fine that there is no value (null/empty) because a value is not required, therefore I am using -5 for »Nothing«. There can be other columns to let us know when a null/empty value represent missing or nothing.

Early arriving fact as an Unknown member (or not yet assigned member)
When a data set is transformed to a fact table, there can be a business key column which value has not yet been received to the related dimension table and therefore does not yet exist. To keep all the fact data rows in the fact table, the related dimension table will first have to insert the new business key value with a »Unknown« value (called an inferred member) which later will be overwritten in the dimension with the correct text value then it is known. The »Unknown« value gets the next dimension key value as a surrogate key identity as a unique sequence number and will be used in fact table like any other dimension key values.

A dimension table can at the same time have several »Unknown« member values with their own business key value, dimension key value, and a text value can include a value of a business key like »Unknown 886«, »Unknown 887«, »Unknown 888« to distinct them for a business user. When a regular dimension has a hierarchy, the levels can have text value »Unknown« as a special branch in the hierarchy.

Late arriving dimension

When dimension data arrives after a related fact has been loaded because dimension data has been delayed:

- If business key value not exists in dimension then insert new row to dimension.
- If business key value exists in dimension as an inferred member »Unknown« from a »Early arriving fact« then update the row by overwrite its values as a correction of data to achieve correct values.
- If business key value exists and the delayed data comes with a business date of valid or a date of change back in time, then a type 2 dimension becomes more complex because timeline metadata columns ValidFrom and ValidTo has to be adjusted, and revisiting a fact table to update the rows dimension key column to refer to the right dimension data at that business date through a new date range lookup. You have to consider if it is allowed to change old data and old reporting result.

Late arriving fact
When fact data has been delayed maybe it is including a date that can be used to carry out a date range lookup on a business key value in a dimension to fetch a dimension key value for the current dimension member at that time when an event occurred, if dimension keeps history, and to add the dimension key value to the fact table row.

You have to consider if it is allowed to add a late arriving fact row because it will change an old report. For example, a manager already got the report of sales for the first quarter and at June 5 a late sales fact for March 31 is arriving and when it is added to the Sales fact, the report for first quarter will change so it does not match the old reporting result.

When building a ETL process for a fact that is using **type 1** dimension we do a business key lookup to fetch a dimension key value to a fact row with this criteria:

```
dim.Business_key = fact.Business_key AND dim.IsCurrent = 1
```

Or using a durable key from a key mapping table:

```
dim. Durable_key = fact. Durable_key AND dim.IsCurrent = 1
```

When building a ETL process for a fact that is using **type 2** or **type 7** or type 4 - 6 dimension we do a business key **date range lookup** to fetch a dimension key value to a fact row with this criteria:

```
dim.Business_key = fact.Business_key AND
fact.TransactionDate >= dim.ValidFrom AND fact.TransactionDate < dim.ValidTo
```

Or using a durable key from a key mapping table:

```
fact.Durable_key = dim.Durable_key AND
fact.TransactionDate >= dim.ValidFrom AND fact.TransactionDate < dim.ValidTo
```

When a coming fact data row is a late arriving fact with a date stamp in a column TransactionDate, a ETL process will use a date stamp to fetch a dimension key value that represent the **registered values** for the fact row.

In a general perspective for fact data and a ETL process is, that it will always use a date range lookup and therefore a late arriving fact is not something special, except that it affects a report. Therefore an audit trail can inform business users about a late arriving fact, read more in section 6.2.2.

Date Range Lookup because of the >= and < (Range Filtering) is an in-memory lookup to assign a dimension key to each coming fact row in a streaming ETL process of rows from a source system to translate and replace a business key value with a dimension key value to be saved into a fact data row in a fact table according to a TransactionDate from a source system.

Other terms for TransactionDate could be EventDate, HappeningDate or OccurredDate in the business sense e.g. OrderDate, PurchaseDate, SalesDate, InsuranceCoverageStartDate, PlanEffectiveDate or ValueDate and of course a time (o'clock) can also be added.

Date columns in a fact table to make sure of match the old reporting result:

- **TransactionDate** when something occured in the business sense.
- **RegisteredDate** when it was known in the system.
- **InsertTime** when it was known in the data warehouse fact table.

When the data warehouse receives a correction to a fact row or a fact row arrives late these dates can be helpful. A manager got a report of sales for the first quarter at April 2. We can mark the report with an ExecuteDatetime 2010-04-02 4pm to be

used when the manager at August 16 wants to recreate an old report as it actually was looked at the time of creation, we can search the fact data:

InsertTime <= ExecuteDatetime.

We can make a new report as it should have looked given corrections made to the data after its creation: InsertTime <= Now.

Early arriving dimension

When a dimension has members that is not yet been referred to from a fact row. There will never be a method of payment called »Blood«, but I still keep it as a member of the dimension.

Summary of Artifact values or Inferred members

Kimball Design Tip #128 and Kimball Design Tip #171 has a variety of constant artifact values or inferred members with fixed key values for a dimension such as:

»Missing« (-1) when a source system provides a business key value that is **null**, meaning it is not present in some data, not registered, not reported or »hasn't happened yet« or »be available later« or »not yet available« because the business key to be determined is expected to be available later and fact table will be updated thereby. Therefore the fact table dimension key column for the dimension gets the value -1 for »Missing value« as a not expected value. A missing value indicates that we would expect a value, which would not always be correct, see later under »Nothing«.
(In danish Mangler, Ikke angivet, Ubestemt, Uoplyst, Ikke forventet).

»Not available« (-2) when a source system provides a business key value that is not known and not found and unavailable in the data warehouse because the value does not exists in the data warehouse e.g. CustomerNumber 424-15-90. Therefore the fact table dimension key column for the dimension gets the value -2 for unknown in a general term for an **unknown** member because it is not all values I like to make as an inferred member for a dimension. A CustomerNumber in a source system could be type-in wrongly and therefore is it not available for a dimension in the data warehouse, and I find it better to mark the dimension in the fact with -2 instead of inserting a wrong value into the Customer dimension. Some wrong CustomerNumbers will in the fact table gets the value -2 and therefore become homeless, but an Audit trail can fetch them and reporting them and a source system can be corrected. In a real-time load it could happen that a sales transaction is referring to a CustomerNumber that is not yet in the Customer dimension and inferred member is the only way out if it is not possible to use a snapshot on a source system to create a read-only and static view as a transactionally consistent with the database as of the moment of the snapshot's creation.
(In danish Data værdien er ikke tilgængelig, Værdien eksisterer ikke og er ikke til rådighed i dimensionen, Ikke defineret, Ukendt).

»Not applicable« (-3) when the dimension is irrelevant for the fact row (N/A, N.A., NOAP or »Irrelevant« or »Not reported at detail level«). It is best to avoid a N/A dimension by making separate fact tables.
(In danish Ikke anvendelig, Ikke relevant, Er ikke krævet, Ikke indberettet på detaljeniveau).

»Wrong«, »Incorrect«, »Bad«, »Corrupt«, »Dirty« (-4) when a source system provides wrong or bad business key value or not enough data to determine the appropriate dimension key identity. This may be due to corrupted data in a source system or incomplete knowledge of the business rules for the source data for the dimension (wrong data).

»Nothing« (-5) when we know that it is fine that there is no value (null/empty) because a value is not required in a source system and therefore is not missing, not expected and therefore a value is not required in the fact table column. There can be other columns to let us know when a null/empty value represent missing or nothing. The dimension gets the value -5 for »No data«. When the date dimension is role-playing for a BirthDate value -5 might stands for »Prefer not to say« or »No answer« or in a general way »No data«. Sometimes it is fine that there is no value (null/empty) because a value is not required in a column, therefore I am using -5 for »Nothing«, and for another column I am using -1 for »Missing«.
(In danish Ingen, Ingen tildelt, Ingenting, Intet, Ikke oplyst, Findes ikke, Blank, Tom, Uden data, Forventet ingen værdi fra tid til anden).

When you know that a product is missing in your invoice line item, it is fine to call it -1 »Missing product« (in danish Mangler produkt). When you know that a customer don't need to have a role, it is fine to call it -5 »No role« (in danish Ingen rolle). For example, a FactClassInstructor has two dimensions for a primary instructor and a secondary instructor and a role-playing DimInstructor. Sometimes a class has no secondary instructor therefore I use dimension key value -5 in the fact row with the term »No instructor« in DimInstructor.

When a type 2 dimension is expanded with new columns the old historical rows do not have a value for the columns, we can use value »Before column« (-9). The

same apply to a fact table expanded with a new dimension key column where we back in time (old fact rows) have no value for that dimension therefore we use dimension key value -9 for »Before column«.

Example of dimension members plus artifact values or inferred members
Religion
Buddhist
Christian - Catholic
Christian - Protestant
Christian - Other
Hindu
Islam
Jewish
Other
No Religion
No Answer
Prefer not to say

Religious values
Very Religious, Religious, Not Religious, No Answer.

Example of artifact values or inferred members
The dimension key values -1 to -7 is not carved in stone, please make the values that fit the dimension, e.g. a FactQuestionnaire has a DimRespondent with following key values in no particular order.

-1	Missing respondent	E.g. a questionnaire has a null for the respondent, that is a data quality issue in the fact row.
-2	Unknown respondent	E.g. a questionnaire is answered by an unknown person, that is a data quality issue in the dimension.
-3	Irrelevant respondent	E.g. a test user is not relevant for the questionnaire N/A.
-4	Wrong respondent	E.g. an mistake in the registration of the person.
-5	Anonymous respondent	E.g. not missing and not unknown person, prefer not to say, I choose to call the person anonymous.
-6	Male respondent	E.g. instead of total anonymous useful for DimGender.
-7	Female respondent	E.g. instead of total anonymous useful for DimGender.

Example of Early arriving fact and Late arriving dimension
A dimension table can have a metadata column like IsInferred as a boolean (bit) with two values True (1) and False (0):

IsInferred = 1 when the row represents a business key value from a source system with no descriptive data values that become an unknown value in data warehouse.
IsInferred = 0 when the row represents a known value in a source system.

Customer dimension example with dimension key _key, business key _bkey and two text values and a metadata column IsInferred to flag or mark the inferred members that has been inserted with default Unknown text value to handle »Early arriving fact«.

Customer_key 134–136 rows is a placeholder until the actual data arrives, the rows are called inferred dimension members and they ensure that fact tables can reference existing dimension keys without any problems.

Customer _key	CustomerId _bkey	Region	Name	IsInferred
-2	-2	Unknown	Unknown	1
-1	-1	Missing	Missing	1
1	176	Europe	Hans Andersen	0
2	359	North America	Joe Watson	0
3	421	Europe	Marie Beaulieur	0
134	**886**	**Unknown**	**Unknown 886**	**1**
135	**887**	**Unknown**	**Unknown 887**	**1**
136	**888**	**Unknown**	**Unknown 888**	**1**
137	889	Asia	Jian Lewai	0
138	890	North America	Emily Gates	0
139	891	North America	Michael Brown	0

Customer dimension is born with two default artifact values or inferred members:

- -1 for missing in case fact data do not have a customer business key (is null).
- -2 for not available in case fact data has an unknown customer business key that by a mistake in a ETL process has not been inserted as an inferred member in customer dimension, therefore -2 helps to ensure a robust solution and handle a lookup failure during a fact table load.

It is normal procedure to overwrite an inferred dimension member value data row when the source system provides values for Region and a Name regardless of dimension type and set IsInferred = 0 to handle »Late arriving dimension«.

It is seldome for type 2 to insert a new row to keep history. By overwrite an inferred dimension member we achieve an improvement of the fact rows that is using the associated dimension key, because the registered values in a dimension will get the correct text value then it is known. (Kimball Design Tip #78).

Kimball page 479 and 516 says, it is okay to do a correction of data as make a type 1 overwrite changes in a type 2 dimension, and set IsInferred to 0 to get correct registered values. I say it is up to you to think about the overwrite, because will it be okay after a week or a month that a report back in time will have a different output in present time?

Kimball says, there is no need for revisiting a fact table for making an inferred member to a normal and correct member of a dimension.

Customer dimension has received corrected values for Customer_key 134–136 and they have overwrite the Unknown values to handle »Late arriving dimension«.

The fact table is untouched or unchanged.

I prefer to have an audit column called CorrectionTime to tell when an inferred member has been corrected by a ETL overwriting process of a row in a dimension.

Customer_key	CustomerId_bkey	Region	Name	IsInferred	CorrectionTime
-2	-2	Unknown	Unknown	1	
-1	-1	Missing	Missing	1	
1	176	Europe	Hans Andersen	0	
2	359	North America	Joe Watson	0	
3	421	Europe	Marie Beaulieur	0	
134	**886**	**South America**	**Carlos Lopez**	**0**	**2007-06-05**
135	**887**	**Australia**	**Olivia Taylor**	**0**	**2008-02-07**
136	**888**	**Africa**	**Ada Musa**	**0**	**2009-08-16**
137	889	Asia	Jian Lewai	0	
138	890	North America	Emily Gates	0	
139	891	North America	Michael Brown	0	

5. Dimensional modeling methodology

Dimensional modeling ends up with a dimensional model as a star schema of one fact and some dimensions or as a constellation schema of few facts and some dimensions. A fact with measures, metrics, analysis variables is surrounded by conformed and shared dimensions with context descriptive data, text, hierarchy, group and band to explain the measurements.

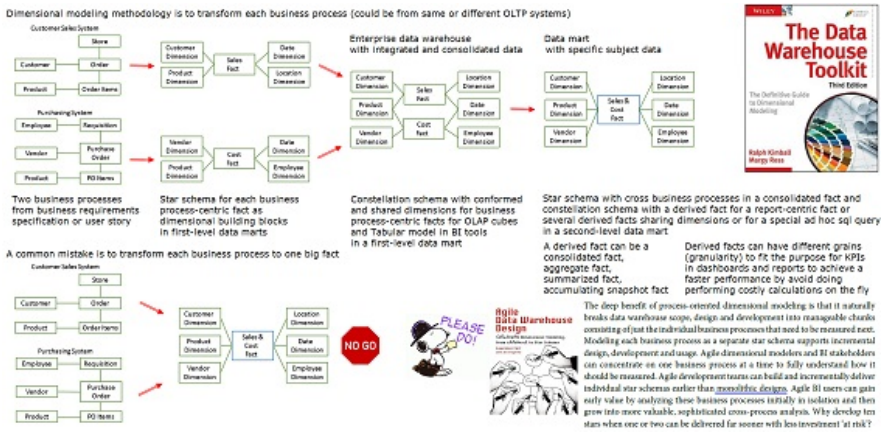
Kimball has two levels of data marts in his second edition from 2002 page 80 and 184, and in The Kimball Group Reader from 2016 page 45-46, 168-69, 273, 414, 468-471, 556, 689-96, and in an article at kimballgroup.com.

First-level dimensional model with a base-level fact in a data mart for dimensional building blocks in a star schema for each business process-centric fact to become a constellation schema with integrated and consolidated data, conformed and shared dimensions.
The focus is on business processes with fact tables at its lowest possible level of data point called atomic grain or most detailed grain available (e.g. transactional fact) for providing analytical queries at the most detailed level.
Sometimes called a core schema with detail level data.

Second-level dimensional model with an aggregate-level fact or a consolidated fact in a data mart from one, two or more first-level data marts to combine and cross business processes in a star schema with consolidated dimensions in a consolidated mart, or in a constellation schema with a derived fact for a report-centric fact or for a special ad hoc sql query with specific subject data as a subject mart for an analytical mart, a reporting mart or a delivery mart.
The focus is on integration of business processes with fact tables that can be at a higher aggregated level (e.g. periodic snapshot fact, accumulating snapshot fact) for providing advanced analytical capabilities.
»Drilling across separate business processes is one of the most powerful applications in a data warehouse«, said Kimball Design Tip #156.
Sometimes called a custom schema with detail level and overall level data.

I wouldn't have forecast and actuals in the same fact in first-level, because data occur at different times for different reasons. In second-level I can have an aggregate fact where data has been converted and equated for comparison purposes.

We must in a first-level data mart avoid a **big wide fact** at all costs by modeling a business process dimensional model. It is great to have first-level data marts with **lean facts** before tackling the task of consolidating and making specific subject data in a second-level data mart.



[Third-level data mart has Enriched Fact Table (EFT), One Big Table (OBT), Analytical Base Table or Analytical Business Table (ABT) that includes descriptive columns from its associated dimension tables, rather than just dimension keys.
For dimension data in a join with a fact we need to choose between current values and registered values to be inserted into the table.]

Kimball page 143-146 talks about a single big wide fact table versus multiple lean fact tables where these observations lead to more fact tables:

- Different source systems, let one source data become one fact table.
- Activities as separate and distinct business processes with transaction types.
- There is unique dimensionality for the various transaction types where several dimensions are applicable to some transaction types but not to other, and not like to have fact rows with dimensions as »Not applicable« (N/A).
- Less complexity in a ETL process for a lean fact table in first-level dimensional model, let complex integration be handled in second-level dimensional model.
- Update frequency of a fact table, e.g. daily for most data and weekly for some special data that has a long ETL process time performance.

Please think dimensionally and not relationally.

An example of measures
In first-level data mart we can calculate measures like Gross Revenue, Returned Revenue, Gross Orders, Fully Returned Orders with the needed business logic for filtering, joins, exclusions, mappings.

In second-level data mart we can calculate measures like:
Net Revenue = Gross Revenue – Returned Revenue
Net Orders = Gross Orders – Fully Returned Orders
Average Order Value (AOV) = Net Revenue / Net Orders

Ask business users
Ask not the business users »what do you want?« in a data warehouse or pull out a list of data elements to determine what's needed.

Ask »what do you do (and why)?« and let the business users tell you about the business process, that you will dimensional modeling as a fact table with a grain and with surrounded dimensions with context descriptive data and with measurements. (Kimball Design Tip #110. A good example in Kimball Design Tip #18).

Dimensional modeling methodology has a four step process from Kimball that is in a business process aligned manner rather than departmentally aligned manner. The four step process is mostly focused on the first-level data mart.

5.1. Four step process
Knowing the business ontology is important.
I have summarized from Kimball page 38-41, 70-77, 300-304, 434-442 and Kimball Design Tip #3 together with my own words.

A **business process** refers to a specific sequence of operational activities or tasks that are performed in a company or in an organization to accomplish and achieve a particular objective or goal such as taking an order, making an invoice, handling a service call, processing an insurance claim, customer interactions, financial transac-

tions, registering students for a class, do a compilation of measurement data from nature, etc. For a dimensional modeler, the business process is an event or activity which generates or collects metrics, measures, measurements or facts.

Kimball said: »Business process data is the foundation upon which we build the data warehouse.«

A business process consists of one **business process event** or several business process events that capture data into a business entity and generate data from a business entity or several business entities with measurements through the use of several operational systems that become source systems for a data warehouse.
A business process event is a point in a business process.
Business process events are things that happen that we want to record and be able to measure and evaluate, and we store it in business entities.

A Data Flow Diagram can be one way to illustrate a business process with events and entities, and combine it with screenshots from the operational systems to fetch terms and their namings for Input data and Output data.

A **business entity** has properties for data and is often implemented in an operational system based on a relational model database as a table or as several tables with relationships, and a table has columns and rows to store data, where a primary key is unique and for use in identification of a row.

Read about business process in Kimball Design Tip #47, Kimball Design Tip #69, Kimball Design Tip #72 and Kimball Design Tip #154.
Other readings in What Is a Business Process? and Regulation-Based Dimensional Modeling for Regulatory Intelligence.

An illustration of an operational system with many business entities with data to select a business process with two business process events for dimensional modeling to a star schema.

Dimensional modeling methodology emphasizes understanding business processes, which can lead to a dimensional model that is highly tailored to specific business needs but may not be flexible enough to accommodate future changes or diverse analytical requirements.

The basic principle in dimensional modeling is that a fact table represents a business process event or state, and dimensions provide context to the fact, therefore with its dimensions the fact table can stand on its own.

If the business process events are independent and have different measures and grains, they need to be in their own fact table.

This allows you to construct a wide enterprise data warehouse in small steps.

Analysis can be performed on a single star schema or combined with other star schemas (across common conformed dimension columns) for more complex analysis. Each new »piece« expands the scope of analysis that can be performed against the data warehouse.

A fact table is a place where we store measurements and metrics that come from a specific business process. Dimension tables describe these measurements and metrics, and they are conformed and shared dimensions that surround a fact table.

A fact table contains facts of a business process, and for the most atomic information captured by a business process. Atomic data is the most detailed information collected; such data cannot be subdivided further, wrote Kimball in his second edition from 2002 page 34. An atomic fact table is representing a particular business process event, and the **grain** of the fact table is at its lowest level of detail as it relates to that specific business event.

We must design a **business process-centric fact** and not take a report-centric approach.

Kimball page 300: »Dimensional models should not be designed solely to deliver specific reports or answer specific questions.«

It is fine to design a report-centric **derived fact** table from a fact table or from multiple fact tables from different business process-centric fact tables. A derived fact can be a consolidated fact, an aggregate fact, a summarized fact, an accumulating snapshot fact, read about types of fact tables in section 3.3.

Conformed dimensions are the integration points between the disparate business processes of an organization, and are ensuring semantic consistency between the processes.

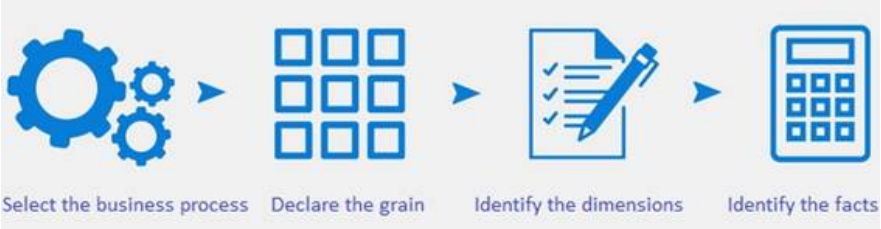
An example of different fact tables for each business process
Healthcare will have an admissions fact because it is a provider and not the payor. A separate lab work fact because we want fact tables to fit a business process, therefore it will not make sense to try to jam admissions and lab work together in one fact table as they are different business processes that occur at different times. Procedures should be in a different fact table from admissions because a procedure is a business process event that occurs at a different grain from the admission. Since procedures represent revenue, we may want to include lab work in the procedure fact as well. Assuming the lab work fact has more information than what we would need in the procedure fact, both fact tables can co-exist with each other. Procedure facts would be used for revenue measures, while the lab work fact would be a subset and contain more specific information germain to the lab work.
Fact tables for admission, lab work and procedure share many conformed dimensions such as patient, diagnosis, admission id (a degenerate dimension), etc.
Diagnosis is treated as a multivalued dimension in all cases.

From Relational model to Dimensional model
Dimensional modeling basic idea is to do a decomposition of a business entity often from a relational model and to make a new composition to a dimensional model where a business entity becomes dimensions and fact to form a data warehouse to preserve when data was happening, occurred, registered or took place in operational systems to form a data warehouse according to the definition.

Dimensional modeling uses facts (measures) and dimensions (context). Facts are typically (but not always) numeric values that can be aggregated, and dimensions have values in hierarchy, group, band with descriptors that define and identify the facts.

Dimensional modeling architecture uses data areas or data layers or data zones with a data flow, as Kimball in his second edition from 2002 page 7 displays in an illustration to show, that a data mart is based on a single business process.

Four step process for dimensional modeling to star/constellation schema



1. Select the business process to model by gathering and combining an understanding of business needs from business requirements specification or user story and available data in source systems. In practice look after a business process event, a transaction, a circumstance, an instance, an incident or a state. Measurements data from a business process event with a business entity is stored in a fact table. Most fact tables focus on the results of a single business process. Multiple business process measures can be included in a single fact table provided that those measures are captured within the same context and level of granularity. The measures must relate at the same transaction level as all other information in the fact table record.

Naming a business process-centric fact table.

2. Declare the grain or level of detail that is related to the business process by describe exactly what a single business process-centric fact table row represents.

What is the scope of the measurements data from a business process event with a business entity to tell »how many«, »how much«, »how often«, »how long« or »how far« to store in several measures.

The grain must be declared before choosing dimensions or facts because every candidate dimension or fact must be consistent with the grain, so they indicate the fact table's granularity.

Atomic grain refers to the lowest level at which data is captured by a given business process. Start by focusing on atomic-grained data because it withstands the assault of unpredictable user queries. Or most detailed grain available.

Later, some rolled-up summary grains are important for performance tuning, but they pre-suppose the business's common questions.

The choice of the fact table grain is one of the first steps in the conception process of a dimensional data model, and therefore essential in the whole conception of a

dimensional data model.

Determination of a type of fact table and its grain is important.
Naming a business process-centric fact table according to its grain and to its type of fact table.
Read about grain in chapter 2 and types of fact tables in chapter 3.

3. Identify the dimensions to match the grain of a business process-centric fact table by context descriptive data from a business process event with a business entity to tell »who, whom, whose, what, which, where, when, why, how« context and to store in several dimension tables with conformed column names.

Determination of a type of dimension table and its type of slowly changing dimension is important.
Naming dimension tables and naming their columns.
Read about types of dimension tables with columns, hierarchy, group, band, business key in chapter 4.

4. Identify the facts to match the grain of a business process-centric fact table by measurements as quantitative numeric measures and metrics data from a business process event with a business entity. Measurement of a business process.

KPI Key Performance Indicator requirements from the business users will appear as a result of a data integration to support data-driven decision-making process. Some measures are from source systems as a result from a business process event, and other measures are additional derived from other fact tables, and other measures are calculated by some business rules.

Determination for each measurement a type of measurements, measures, metrics.
For example, a sales amount, a duration of an event, an average length of time and a count of occurrences.
Naming each measurement and determining a unit for a measurement.
Read about types of measurements, measures, metrics in section 3.2.

Dimensional modeling principle
The dimensional modeling principle is that a business process will represent a fact table and a business process event with a business entity will represent a measurement in the fact table with context by descriptive data values in surrounding dimension tables to business users in different business functions and departments in a company or an organization.

Several business processes ends up in several facts surrounded by conformed and shared dimensions displayed as a star schema or a constellation schema diagram in several data marts in a data mart area for a data warehouse solution.

At the end update the Enterprise Data Warehouse **BUS Matrix** to capture a relationships between the business processes and the dimensions in a business process matrix. It is an Enterprise Data Warehouse BUS Architecture approach with a business matrix. BUS stands for Business User Support.
Illustration First-level and Second-level illustration With types of fact tables

A BUS matrix is usually represented as business processes versus dimensions which I see as a logical model, and I see a physical model where a BUS matrix is represented as facts versus dimensions, and facts can be divided in business processes for a constellation schema, sometimes referred to as a Constellation matrix.

In process-based management, where processes are transversal to departments, the organizations can be represented by a matrix with processes versus departments. Read about matrix part 1 and matrix part 2 and matrix part 3.

There is one business process for making a budget, another for making a forecast and another for actual and the grain and dimensionality will be different. If this is General Ledger (GL) data (account balances, not journals), then you should be able to use a single fact with a ledger type dimension (actual, budget(s), forecast(s)). If you are maintaining journal data (actuals only) this would be in its own fact. If you have both, the journal fact should include the ledger type dimension so you can easily reconcile the journal facts with the balances.

Sometimes there are multiple potential designs and it may be necessary to think them all through carefully to decide which is best.

In addition to Kimball with many descriptions, for example from IBM
Data Warehousing Architectures
Dimensional modeling guide
Dimensional data modeling
Seven steps for successful data warehouse projects

Four step process explained in an English video by a Danish teacher

What process are you trying to represent?
What are the actions/events/states that occur during the process?
What is the context for these actions/events/states?

Read about the dimensional modeling process

Please, remember a good review process of your dimensional model, read Kimball Design Tip #108.

Extensions such as Puppini bridge can be read in book Unified Star Schema, here is two illustrations a bridge between two facts and the contents of a bridge.

Medallion architecture has three layers: (Olympic Games architecture) Illustration
- Bronze (Raw data): Source system provided data as it is, stores data in its original, raw format as it is ingested from various sources. It serves as the historical, immutable record of the data. For example, Delta - parquet tables.
- Silver (Validated data): Data from the bronze layer is cleansed, conformed, validated, and enriched. It becomes more reliable and is structured for querying and analysis. First-level dimensional model with a base-level fact in a data mart.
- Gold (Enriched data): Data from the silver layer is curated and containing highly refined data that is aggregated and optimized for specific business use cases like business intelligence, reporting, and machine learning. Second-level dimensional model with an aggregate-level fact or a consolidated fact in a data mart.

5.2. Invoice example

We have a sales business process for customer invoicing and a business process event for making a sales invoice to a customer for a purchase of a product or a service with data from a business entity in an operational system based on a relational model where an invoice business entity has two parts of data.
Sample of a sales invoice to a customer:

East Repair Inc.
1912 Harvest Lane
New York, NY 12210
Sales rep: Ole Ritter

INVOICE

Bill To	Ship To		
John Smith 2 Court Square New York, NY 12210	John Smith 3787 Pineview Drive Cambridge, MA 12210	**Invoice #** **Invoice Date** **P.O.#** **Due Date**	US-001 11/02/2019 2312/2019 26/02/2019

QTY	DESCRIPTION	UNIT PRICE	AMOUNT
1	Front and rear brake cables	100.00	100.00
2	New set of pedal arms	15.00	30.00
3	Labor 3hrs	5.00	15.00
		Subtotal	145.00
		Sales Tax 6.25%	9.06
		TOTAL	**$154.06**

An invoice is basically composed of one invoice header and several invoice items:
- Invoice header: store, sales rep, customer, ship-to address, bill-to address, invoice no, invoice date, due date, sales amount before tax (subtotal), sales tax percentage rate, sales tax amount, sales amount after tax (total).
- Invoice line item: invoice line no, product (or service), quantity, unit price, sale amount.

There is no business process events for discount, promotion, coupon, etc.

Four step process for dimensional modeling of an invoice to a dimensional model

1. Select the business process as a Customer sales invoicing, that refers to a business process event to generate data in a source system for a customer sales invoice when a product or several products were sold in one of the company's stores.

An invoice business entity has a header with multiple line items. It could become a »parent invoice header fact« table with 6 dimensions and 4 measures and become a »child invoice line item fact« table with 1 dimension and 3 measures.

Kimball Design Tip #25: »We can't roll up our business by product! If we constrain by a specific product, we don't know what to do with invoice level discounts, freight charges and tax.« source.

Kimball Design Tip #95 Transaction Line Fact from parent/child data model source.
The Kimball Group Reader from 2016 page 304-307 Managing Your Parents.
Kimball page 59 Header/Line Fact Tables, page 60 Allocated Fact, page 181-188 Header/Line Pattern to Avoid and Invoice Transactions.

Dimensional modeling technique will take the parent invoice header dimensions and measures and allocate them down to the child invoice line item fact table (or sales invoice line item fact). Please, don't create a Header fact table.

An invoice is an business event, a fact, and the context of that event are in dimensions related to the invoice.

The result is a Sales invoice business process-centric fact table, because the company has also a business process for supplier purchase invoicing that one day can be a PurchaseInvoice fact. In short we get a **SalesInvoice fact table** with a primary key: InvoiceNumber + InvoiceLineNumber or InvoiceNumber + Product dimension key since a product can only appear once in an invoice.

2. Declare the grain as a Sales invoice item with one product.
There is often many products in an invoice business entity, which gives many invoice line items for an invoice. There is often many invoices or invoice business entities in one day.

SalesInvoice fact grain is that a single fact table row represents an invoice item that is a product in an invoice line in a sales invoice to a customer.

[The grain of the fact table SalesInvoice is a sales invoice line item with a product on each invoice made to one of the customers.]

Each fact table row is an individual sales invoice item with an invoice line number. Of course, an invoice can have several products in several invoice lines, which are represented in several rows in the SalesInvoice fact table with the same value in columns for invoice number and purchase order number.
The total Sales tax amount will be allocated to each product. It is called allocating.
The SalesInvoice fact will be a transactional fact with a transaction grain.

3. Identify the dimensions as descriptive data values:
- Customer with name and address
- Product with number, label, brand, category, unit price
- Store with code, store name, address
- Sales rep (employee role-playing)
- Ship-to address (address role-playing and outrigger to Customer)
- Bill-to address (address role-playing and outrigger to Customer)
- Invoice date (date role-playing)
- Due date (date role-playing)
- InvoiceNumber (DD) and
- PurchaseOrderNumber (DD) will be repeated for each row when an invoice contains more than one invoice items because of several products
- InvoiceLineNumber (DD) is unique for each invoice line

There is no Invoice dimension because all data of an invoice is in SalesInvoice fact.
Can be extended with dimensions for promotion with discount and payment term.

4. Identify the facts as a Sales invoice item with one product with measures:
- Quantity (fully-additive) of a product (number of units)
- Unit price (non-additive) of a product, current copy from Product dimension
- Sales amount before tax (fully-additive) (Quantity x Unit price)
- Sales tax percentage rate (non-additive)
- Sales tax amount (fully-additive)
- Sales amount after tax (fully-additive)
- TotalSalesTaxAmount is a sum of Sales tax amount for an invoice
- TotalInvoiceAmount is a sum of Sales amount after tax for an invoice

At detail level will analysis of products and customer data be carried out for the purpose of marketing planning, etc.

TotalSalesTaxAmount and TotalInvoiceAmount has a value for each invoice line, therefore they become (non-additive). Some data modelers has a value only at the first invoice item and the other invoice items has value 0 to achieve fully-additive measures for a dashboard to display an overall level of summary total invoice amount per day, per week or per year divided by customers residence cities, but the values can not be slicing by product, brand or category.

The SalesInvoice fact table is a base-level fact with detail level data in a **first-level** dimensional model without TotalSalesTaxAmount and TotalInvoiceAmount.

A **second-level** dimensional model can have a **derived fact** as an aggregate-level fact that derives data and do a computed summary of measures from a base-level fact in a first-level data mart, e.g. TotalSalesInvoice fact table with grain of an invoice where InvoiceNumber becomes unique for each row with measures TotalSalesTaxAmount and TotalInvoiceAmount and without the Product dimension.

TotalSalesInvoice fact grain is that a single fact table row represents a sales invoice to a customer.

The number of invoices from the two facts with different grain:

- Select Count(*) or Count(InvoiceNumber) From TotalSalesInvoice fact table.
- Select Count(Distinct InvoiceNumber) From SalesInvoice fact table. (DistinctCount)

When we want SalesTaxAmount and InvoiceAmount of brand and category levels, we can have another aggregate-level fact with a shrunken dimension of Product dimension with Brand and Category.

The number of facts in a second-level dimensional model dependents on the use of data, large data volumes and the performance / response time in dashboards and reports.

Examples of other business processes for customer
A business process for Customer payment refers to a business process event of ensure an invoice payment status business entity to a SalesInvoicePayment fact with invoice total amount, invoice payment amount, invoice remaining amount, payment date, payment method, payment status (paid/unpaid/settlement).

A business process for Customer shipment refers to a business process event of delivery the products to a Shipment fact.

A business process for Customer promotion refers to a business process event of offer a discount and coupon to a product in a store to a Promotion fact.

A business process for Customer order refers to business process events of make an order from a customer to an Order fact for two departments of sales and marketing.

A business process for Customer purchasing refers to a business process event of sale a product or a service to a customer purchase business entity to a Sales fact with a unit cost price and unit sales price to calculate a gross profit amount and a gross margin as gross profit amount / revenue amount (non-additive).

A business process for Customer service request refers to a business process event when a customer contacts and asks for service to a Service request fact.

A business process for Customer satisfaction refers to a business process event of feedback as comments and complaints from a customer to a Feedback fact.

Examples of business processes for product
A business process for manufacturing of products to a ProductManufacturing fact.
A business process for inventory of products to a ProductInventory fact.

Examples of a business processes for store
A business process for renting of stores to a StoreRent fact.
A business process for employment in a store to a StoreEmployment fact.

Examples of a business processes for employee (sales rep)
A business process for competencies and courses of employees to a EmployeeCompetence fact.
A business process for sick days and vacation of employees to a EmployeeHealth fact.

5.3. Kimball 34 subsystems of ETL

Kimball page 449-496 describes 34 subsystems of ETL with a good summary of the dimensional modeling methodology with these headlines:

1. Data Profiling
2. Change Data Capture System
3. Extract System
4. Data Cleansing System
5. Error Event Schema
6. Audit Dimension Assembler
7. Deduplication System
8. Conforming System

9. Slowly Changing Dimension Manager
10. Surrogate Key Generator
11. Hierarchy Manager
12. Special Dimensions Manager
13. Fact Table Builders
14. Surrogate Key Pipeline
15. Multivalued Dimension Bridge Table Builder
16. Late Arriving Data Handler
17. Dimension Manager System
18. Fact Provider System
19. Aggregate Builder
20. OLAP Cube Builder
21. Data Propagation Manager
22. Job Scheduler
23. Backup System
24. Recovery and Restart System
25. Version Control System
26. Version Migration System
27. Workflow Monitor
28. Sorting System
29. Lineage and Dependency Analyzer
30. Problem Escalation System
31. Parallelizing/Pipelining System
32. Security System
33. Compliance Manager
34. Metadata Repository Manager

5.4. Kimball naming example

Dimensional modeling ends up with a dimensional model implemented in a in relational database with tables for dimensions and facts, and with one-to-many relationships between dimension keys as primary keys and foreign keys, meaning it becomes a Entity Relationship data model where entities are labelled with a Kimball naming. (Labelling in danish at lave en mærkning og sætte en etikette på)

An entity or table can have a normal name e.g. Product and Sales, and they can be placed into multiple database schemas based on a Kimball naming to labelling each table in a dimensional modeling, for example:

Dim
Snapdim
Snowdim
Junkdim
Pcdim
Minidim
Bridge
Fact
Lessfact
Sumfact
Tnsfact
Ctpfact
Dvdfact
Hisfact
Snapfact

Views can be placed in their own database schemas dims and facts with a naming to labelling the data it represents such as a role or a type as a suffix, for example:

dims.Client_Current
dims.Client_Current_Unique
dims.ClientRole_Current
dims.Hardware_Current
dims.Software_Registered
dims.SoftwareRole_Current
facts.Client_Current
facts.Client_SnapshotDataEntryAction
facts.ClientRole_Current
facts.ClientRole_SnapshotDiscrepancy
facts.ClientSoftware_Current_Factless
facts.ClientSoftware_DerivedBlockedSoftwareCombination
facts.ClientSourceSystem_Historical_Factless
facts.ClientHardware_SnapshotPeriodic_Weekly
facts.BankAccount_Transactional
facts.Sales_SnapshotPeriodic_Monthly_Summarized

Each data modeler uses different labels for a name of a table, e.g.

Customer_dim	Sales_fact
DimCustomer	FactSales

Dim_Customer	Fact_Sales
dim.Customer	fact.Sales

Extra database schemas for Usage supporting tables in a ETL process:
Keymap for mapping tables between multiple source systems e.g. business keys.
Map for mapping data to a dimension or to a hierarchy, group or band.
Rule for rule tables for enrichment of dimensions, search criteria, facts.

5.5. Kimball's Healthcare example

The most important of all is a good table name e.g. in Kimball's Healthcare example he has tables like:

Health Care Billing Fact (grain item)
Surgical Events Transaction Fact
Medical Record Entries Fact
Hospital Dimension
Patient Dimension
Diagnosis Group Dimension
Diagnosis Group Bridge
Diagnosis Dimension
Treatment Dimension

5.6. More readings about dimensional modeling and other topics

Some of my posts at LinkedIn

https://www.kimballgroup.com
http://www.wikipedia.org/wiki/Data_warehouse
https://www.kimballgroup.com/wp-content/uploads/2013/08/2013.09-Kimball-Dimensional-Modeling-Techniques11.pdf (a good introduction)
http://www.kimballgroup.com
https://www.kimballgroup.com/about-kimball-group/
https://www.kimballgroup.com/data-warehouse-business-intelligence-resources/kimball-techniques/dimensional-modeling-techniques/
https://www.kimballgroup.com/data-warehouse-business-intelligence-resources/kimball-techniques/dimensional-modeling-techniques/star-schema-olap-cube/
https://www.kimballgroup.com/2009/05/the-10-essential-rules-of-dimensional-modeling/
https://www.kimballgroup.com/2007/10/subsystems-of-etl-revisited/ (1-34)
https://www.kimballgroup.com/1997/08/a-dimensional-modeling-manifesto/
https://www.kimballgroup.com/2003/03/declaring-the-grain/
https://www.kimballgroup.com/2003/01/fact-tables-and-dimension-tables/
https://www.kimballgroup.com/2004/10/fables-and-facts/
http://www.kimballgroup.com/wp-content/uploads/2012/05/Fables_Facts.pdf
https://www.kimballgroup.com/2004/03/differences-of-opinion/
https://www.kimballgroup.com/2002/09/two-powerful-ideas/ (data staging area)
https://www.kimballgroup.com/2004/01/data-warehouse-dining-experience/
http://www.wikipedia.org/wiki/Slowly_changing_dimension
http://www.kimballgroup.com/2015/12/design-tip-180-the-future-is-bright/
https://www.kimballgroup.com/data-warehouse-business-intelligence-resources/
http://www.kimballgroup.com/wp-content/uploads/2007/10/An-Architecture-for-Data-Quality1.pdf
https://www.kimballgroup.com/2009/10/six-key-decisions-for-etl-architectures/
https://kimballgroup.forumotion.net/ (thousands of cases and examples)
http://www.1keydata.com/datawarehousing/datawarehouse.html
LeapFrogBI videos:https://www.youtube.com/watch?v=cwpL-3rkRYQ&list=PLrbIyvYCdg0iAUQoxG5vI_yKqzZ2AcgGe
ETL Architecture In-Depth - Dimensional Modelling 101 videos:
https://www.youtube.com/watch?v=DspXXZrSVRk
A danish teachers presentations in English, texts and videos:
http://astridhanghoej.dk/dataanalyticsinfrastructure/
SCD type 2 video: https://www.youtube.com/watch?v=ppASBJ7zs88
https://www.empiredatasystems.com/data-warehouse-concept.html
https://www.vertabelo.com/blog/dimensions-of-dimensions-a-look-at-data-warehousings-most-common-dimensional-table-types/
https://www.linkedin.com/pulse/kimball-still-relevant-modern-data-warehouse-simon-whiteley/
https://www.jamesserra.com/archive/2014/12/the-modern-data-warehouse/
https://www.databasejournal.com/features/mssql/modern-data-warehouse-design-pattern-part-i.html
https://maximebeauchemin.medium.com/functional-data-engineering-a-modern-paradigm-for-batch-data-processing-2327ec32c42a
https://learn.microsoft.com/en-us/sql/relational-databases/track-changes/about-change-data-capture-sql-server
https://learn.microsoft.com/en-us/fabric/data-warehouse/dimensional-modeling-dimension-tables
https://learn.microsoft.com/en-us/power-bi/guidance/star-schema

https://dwbi1.wordpress.com/2010/02/24/primary-key-and-clustered-index-on-the-fact-table/
https://www.astera.com/type/blog/data-warehouse-concepts/
http://moi.vonos.net/programming/dwh-datavault/
https://www.just-bi.nl/talk-about-data-vault-2-0/
http://www.redbooks.ibm.com/
https://en.wikipedia.org/wiki/Data_lineage
Materialized View Concepts and Architecture and Azure Create Materialized View
Steps to Modeling the EDW video:
https://www.youtube.com/watch?v=TjCb34JaVMs
https://estuary.dev/enterprise-data-warehouses/
https://medium.com/@piethein/the-extinction-of-enterprise-data-warehousing-570b0034f47f
https://fastercapital.com/keyword/data-integration.html
https://fastercapital.com/topics/accuracy,-completeness,-and-consistency.html
Data Mart from 1993:
http://www.marcdemarest.com/marts.html
ER diagram 1976:
Peter Chen: The Entity-Relationship Model-Toward a Unified View of Data, March 1976
Agile:
http://www.agiledata.org/essays/dataModeling101.html
Data Mesh:
https://martinfowler.com/articles/data-monolith-to-mesh.html
https://martinfowler.com/articles/data-mesh-principles.html
https://www.linkedin.com/pulse/abn-amros-data-integration-mesh-piethein-strengholt/
https://pragmaticdataorg.wordpress.com/2025/02/17/building-a-data-mesh-with-dbt-and-the-pragmatic-data-platform/
Data architecture:
https://www.ibm.com/think/topics/data-architecture
https://thedataecosystem.substack.com/p/issue-25-role-of-data-archtitecture
Vincent Rainardi articles:
https://dwbi1.wordpress.com/
Nicholas Galemmo articles:
https://dataisdimensional.com/
User story
Guidelines for good code (I prefer easy code to read and maintain with nice comments and arguments and reasons and link to a wiki for more info, diagram and user story).

Books
Ralph Kimball and Margy Ross: The Data Warehouse Toolkit, The Definitive Guide to Dimensional Modeling, Third Edition, 2013. First edition was published in 1996 and was introducing the star schema as the basic solution for modeling multidimensional data and had subtitle: Practical Techniques for Building Dimensional Data Warehouses. Star schema or constellation schema is useful for a Tabular model.
More about the book

Ralph Kimball and Joe Caserta: The Data Warehouse ETL Toolkit, Practical Techniques for Extracting, Cleaning, Conforming, and Delivering Data, 2004.
Page 215 shows a pipeline for a ETL process to lookup key values in dimensions.
Read how to implement a pipeline with date range lookup for SSIS in section 11.4

Ralph Kimball and Margy Ross: The Kimball Group Reader: Relentlessly Practical Tools for Data Warehousing and Business Intelligence Remastered Collection, 2016 including all Kimball Design Tip #1 to #180. Some of the tips and more tips or Google: "Kimball Design Tip #1" or "Kimball Design Tip #2", etc. or download.

Ralph Kimball, etc.: The Data Warehouse Lifecycle Toolkit, 2008.

Nicholas Galemmo, etc: Mastering Data Warehouse Design: Relational and Dimensional Techniques, First edition, 2003.

William H. Inmon: Building the Data Warehouse, Fourth Edition, 2005, first edition published in 1990 to be considered as the birth of a data warehouse definition:
»A subject-oriented, integrated, time-variant, and non-volatile collection of data in support of management's decision making process.«

William H. Inmon and Francesco Puppini: The Unified Star Schema: An Agile and Resilient Approach to Data Warehouse and Analytics Design, 2020.

Dan Linstedt and Michael Olschimke: Building a Scalable Data Warehouse with Data Vault 2.0, 2015.

6. Dimension and fact examples with sample data values
Different implementations of a customer dimension as snowflake, type 1, type 2, type 7, looking more into type 7 and looking at slowly changing facts examples.

6.1. Customer snowflake dimension

Customer table in a source system with CustomerId as a primary key (selected as business key for the data warehouse because the value is immutable and unchangeable) and CustomerNumber build of the nationality (country) and the gender is a mutable and a changeable value) because a customer can change gender).

Customer table from a source system with business key column CustomerId.

CustomerId	CustomerNumber	CustomerName
176	DK-0912-M	Hans Andersen
359	US-3942-M	Joe Watson
421	FR-1429-F	Marie Beaulieur
422	FR-1430-M	Mathis Leon

Every time a customer is inserted into a Customer dimension it will provide with a **dimension key** value in column Customer_key as an artificial auto-generated unique sequence number or an identity column as a surrogate key for the primary key for the dimension for having independence of the business key.

Customer dimension has two artifact values or inferred members for Missing and Unknown, which has been assigned two fixed values -1 and -2 in dimension key column Customer_key at the time of create the customer dimension table.

Customer dimension is a type 1 as a snowflake dimension.

Customer_key	CustomerId	Customer Number	Customer Name	Country_key	Gender_key	Payment_key
-2	-2	Unknown	Unknown	-1	-1	-1
-1	-1	Missing	Missing	-1	-1	-1
1	176	DK-0912-M	Hans Andersen	1	1	2
2	359	US-3942-M	Joe Watson	6	1	3
3	421	FR-1429-F	Marie Beaulieur	3	2	3
4	422	FR-1430-M	Mathis Leon	3	1	3

From CustomerNumber I have derived two extra columns for Country and Gender and they become snowflake dimensions and can be used together with other dimension tables and fact tables. Country_key and Gender_key is snowflake keys.

Payment is taking from another table in the CRM system and becomes a dimension itself and is merged into the Customer dimension to do an enrichment of the customers. Payment_key is a snowflake key to a Payment dimension.

Country dimension has been enriching inside the data warehouse by the business users with ISOcode and Currency values to fulfill reporting requirements. Each country is placed in a continent hierarchy (Africa, Antarctica, Asia, Australia, Europe, North America and South America) in case a report wants a continent overview of the customers and sales.

Country dimension is a type 1 as an Outrigger dimension.

Country_key	Country	ISOcode	Currency	Continent
-1	Missing	--	Missing	Missing
1	Denmark	DK	Danish Krone	Europe
2	England	UK	British Pound	Europe
3	France	FR	Euro	Europe
4	Germany	GE	Euro	Europe
5	Norway	NO	Norwegian Krone	Europe
6	USA	US	United States Dollar	North America
7	China	CH	Renminbi	Asia

Gender dimension is a type 0 as an Outrigger dimension.

Gender_key	Gender	Sign
-1	Missing	◦
1	Male	♂
2	Female	♀
3	Transgender	⚧

THE 71 GENDER IDENTITIES ACCORDING TO FACEBOOK

Payment dimension is a type 1 with an extra data warehouse column to specify the sort order of payments for reporting purpose and a group with groupings.

Payment_key	PaymentMethod	PaymentSortorder	PaymentGroup
-1	Missing	99	Missing
1	Bank account	3	Transfer payment

2	Cash	1	Cash payment
3	Credit card	2	Online payment
4	PayPal	5	Online payment
5	Bitcoin	4	Virtual currency

For a OLAP cube for multidimensional model or tabular model loading data from a Customer dimension, there will be implemented a **view** that joins the relevant tables and with several columns and hierarchies to make an easy search, zoom and drill down to specific data values in a data access BI tool like Excel, Power BI, Tableau, QlikView, Qlik Sense, Alteryx or Targit:

Customer
- Number
- Name
- Gender
- Nationality (Country dimension renamed to be a role-playing dimension)
- PaymentMethod

In terms of granularity I want to see daily sales by customer and product for each individual employee which gives the grain declaration of date, customer, product, employee. The time-of-day from a source system is left out so quantity has been summarized to date level in case a customer purchases the same product at the same employee in the one day, e.g. 10.15am and 3.45pm.
An example of a sales statistic would be: »What is the amount of sale to female customers from France that have been paid with credit card in April 2014?«

Sales fact with dimension key columns to dimensions and columns with measures.

PurchaseDate_key	Customer_key	Product_key	Employee_key	Quantity	Sale Amount
2014-04-23	2	21	12	1	125
2014-04-23	2	15	12	3	75
2014-04-24	3	1	4	5	10
2014-04-25	1	15	12	1	25
2014-04-29	3	14	15	2	35
2014-04-29	4	3	15	3	99
2014-04-30	-1	15	-1	1	0

The last row shows a unwanted sale because it is a stolen copy of a product, therefore the customer is missing and gets key value -1 together with the missing employee and sadly no money into the cash register.

PurchaseDate_key is a role-playing date dimension with the role »purchase«.

A report want to show the total sale amount per gender including Transgender even though there is no transgender customer yet. Since snowflaking has created a Gender dimension with all genders independent of customer and sales, a query will use an outer join to include all genders:

```
SELECT g.Gender, TotalSaleAmount = ISNULL(SUM(s.SaleAmount),0)
FROM fact.Sales s
      INNER JOIN dim.Customer c ON c.Customer_Key = s.Customer_key
      RIGHT OUTER JOIN dim.Gender g ON g.Gender_Key = c.Gender_key
GROUP BY g.Gender_key
```

Gender	TotalSaleAmount
Missing	0
Male	324
Female	45
Transgender	0

When **snowflaking is not wanted** for the Customer dimension, it will contain all the relevant descriptive data columns:

Customer_key
CustomerId
CustomerNumber
CustomerName
Country
ISOcode
Currency
Continent
Gender
PaymentMethod
PaymentSortorder
PaymentGroup

The Customer dimension will contain redundant data e.g. a PaymentSortorder value is repeated for all the customers that is using same payment method. One day

business ask for a new sort order of payment methods because it will be more suitable for the reporting.
I recommend to have a Usage supporting database in the data warehouse with tables where business users can update data through an application. With a Payment table with columns PaymentMethod, PaymentSortorder and PaymentGroup a business user can change the values in column PaymentSortorder. The Usage supporting database will be a data source to the data warehouse where a ETL process will fetch the Payment table to the Input data area and the ETL process will update the Customer dimension table in the Data mart. Since the business does not care of old sort order of payment methods, the column PaymentSortorder in Customer dimension will be treated as a type 1 column meaning a simple overwrite of all the rows in the dimension table.
In the Usage supporting database a business user can add extra data to the countries and hereby enrich the reporting with data that don't exists in the source systems.
Since there is no customer of Transgender, the value will not exists in the Customer dimension and therefore Transgender can't been shown in a report except when it is added through a Union sql statement but data belongs in a row in a table.

When **all dimension data are wanted as fact**, a fact table needs to include extra dimensions for Gender, Country and Payment and drop them from Customer. We get »capture a relationship in the fact«.

6.2. Type 1, type 2 and type 7 with most recent current values

In a source system for customers and their purchases each customer has their own immutable unique number value as a primary key column with name CustomerId. CustomerId becomes a business key in a Customer dimension in the column CustomerId_bkey. (CustomerId could have been added to a key mapping table that generates an artificial auto-generated unique sequence number or an identity column as a durable key for later use in a dimension to glue multiple rows together in a timeline for a type 2 dimension or a type 7 dimension).

A customer has an CustomerId 421 in a source system and her maiden name is Marie Beaulieur. She is inserted into a Customer dimension that keep CustomerId as a **business key** in column CustomerId_bkey.

Every time a customer is inserted into a Customer dimension it will provide with a new **dimension key** value in column Customer_key as an artificial auto-generated unique sequence number or an identity column as a surrogate key for the primary key for the dimension for having independence of the business key.

Customer dimension has two artifact values or inferred members for Missing and Unknown, which has been assigned two fixed values -1 and -2 in dimension key Customer_key either at the time of creating the customer dimension table or at an initial insert or an insert in each ETL process that will check if inferred members exists already else insert them with the identity column is temporarily turned off.

Marie Beaulieur got married at 2007-11-16 and took her new husband's surname, therefore her name is changed to Marie Lesauvage.

6.2.1. Type 1 dimension

When the dimension is a type 1, Marie's maiden name will be overwritten and will be forgotten.

Before Marie changes her name in a Customer dimension type 1.

Customer_key	CustomerId_bkey	CustomerName
-2	-2	Unknown
-1	-1	Missing
1	176	Hans Andersen
2	359	Joe Watson
3	421	Marie Beaulieur

After Marie changes her name in a Customer dimension type 1.

Customer_key	CustomerId_bkey	CustomerName
-2	-2	Unknown
-1	-1	Missing
1	176	Hans Andersen
2	359	Joe Watson
3	421	Marie Lesauvage

The type 1 dimension will always show a customer's current name (most recent name). A Customer_key column will be in a fact table and in a data access BI tool like Excel, Power BI, Tableau, QlikView, Qlik Sense, Alteryx or Targit the fact column Customer_key will be joined to dimension key column Customer_key inside the BI tool's data model.

Sales fact with dimension key columns to dimensions of type 1.

PurchaseDate_key	Customer_key	Product_key	Quantity	SaleAmount
2005-11-10	1	11	1	45
2005-11-10	2	14	2	140
2005-12-01	3	19	2	200
2010-06-30	3	17	1	65
2014-05-08	3	19	3	300
2014-08-16	3	11	2	90
2016-03-15	3	14	1	70
2016-03-15	3	19	1	100
2016-03-31	-1	11	1	0
2016-04-01	-2	-2	2	250

For each row in the fact table, the key value in column Customer_key is pointing to a row in the dimension table and give the current name and most recent name of the customer. Customer_key column in fact table represents **as-is** when it is joined to the dimension table.

At PurchaseDate 2016-03-31 there is a missing customer with key value -1 because a product has been stolen and no money into the cash register.

At PurchaseDate 2016-04-01 there is an unknown customer with key value -2 because a IT software upgrade failed to register a new customer who purchased a new product. Therefore both business keys is not known in the dimensions.

6.2.2. Type 2 dimension

When the dimension is a type 2, Marie's maiden name will remain and will be remembered and her changed name will continue in an extra row. To keep track of when a value is changed, there is two timeline metadata columns called ValidFrom and ValidTo.

ValidFrom and ValidTo represents the span of time when a row in a dimension was the »current truth« in an active period.

ValidFrom and ValidTo columns store the effective dates of validity for a row in a table, e.g. a type 2 dimension table where beginning of time in ValidFrom represents the initial row with 1900-01-01 and end of time in ValidTo represents the current row with 9999-12-31.

The first entering will be giving a ValidFrom as 1900-01-01 for a beginning of time and a ValidTo as 9999-12-31 for a end of time.

Marie Beaulieur as her maiden name was valid from she was born or from she becomes a customer and that date is known in a source system, and she is inserted into a first row in the dimension table with ValidFrom as 1900-01-01 for a beginning of time and with ValidTo as 9999-12-31 for a end of time.

Marie Beaulieur's marriage at 2007-11-16 will create a new row in the dimension table with the ValidFrom 2007-11-16 because she took her husband's surname to Marie Lesauvage, and the previous row will get ValidTo 2007-11-16 meaning that until that date she kept her maiden name.

Dimension key Customer_key is an artificial auto-generated unique sequence number or an identity column as a surrogate key for the primary key for the dimension for having independence of the business key.

Business key CustomerId_bkey is the »glue« that holds the multiple rows together.

Before Marie changes her name in a Customer dimension type 2.

Customer_key	CustomerId_bkey	CustomerName	ValidFrom	ValidTo
-2	-2	Unknown	1900-01-01	9999-12-31
-1	-1	Missing	1900-01-01	9999-12-31
1	176	Hans Andersen	1900-01-01	9999-12-31
2	359	Joe Watson	1900-01-01	9999-12-31
3	421	Marie Beaulieur	1900-01-01	9999-12-31

After Marie changes her name in a Customer dimension type 2.

Customer_key	CustomerId_bkey	CustomerName	ValidFrom	ValidTo
-2	-2	Unknown	1900-01-01	9999-12-31
-1	-1	Missing	1900-01-01	9999-12-31
1	176	Hans Andersen	1900-01-01	9999-12-31
2	359	Joe Watson	1900-01-01	9999-12-31

3	421	Marie Beaulieur	1900-01-01	2007-11-16
4	421	Marie Lesauvage	2007-11-16	9999-12-31

Kimball page 508 calls it no gap exists between rows of the same business key value, meaning that the old current row ValidTo column and the new current row ValidFrom column are using the same datestamp value as we saw above, where Marie changed her surname from Beaulieur to Lesauvage at 2007-11-16.

ValidFrom represents »from now on« (in danish »fra og med«).
ValidTo represents »to and not included« (in danish »til og ikke medtaget«).

What happen to the date when Marie becomes a customer at 2005-12-01? We can add a column called CustomerBeginDate, or we can add two rows at the first time a customer is entering, where ValidFrom in the first row gets date of beginning as 1900-01-01 and ValidFrom in the second row gets date of customer membership as a business date.

Customer_key	**CustomerId_bkey**	**CustomerName**	**ValidFrom**	**ValidTo**
-2	-2	Unknown	1900-01-01	9999-12-31
-1	-1	Missing	1900-01-01	9999-12-31
1	176	Hans Andersen	1900-01-01	1997-04-05
2	176	Hans Andersen	1997-04-05	9999-12-31
3	359	Joe Watson	1900-01-01	2000-01-12
4	359	Joe Watson	2000-01-12	9999-12-31
5	421	Marie Beaulieur	1900-01-01	2005-12-01
6	421	Marie Beaulieur	2005-12-01	2007-11-16
7	421	Marie Lesauvage	2007-11-16	9999-12-31

When we want to know the number of customers at the end of the year 2000:

```
SELECT COUNT(*) AS NumberOfCustomer
FROM dim.Customer
WHERE ValidFrom != '1900-01-01' AND
        ValidFrom <= '2000-12-31' AND ValidTo > '2000-12-31'
```

Based on Kimball Design Tip #11.
The above table is not used in the following example.

Marie Lesauvage remarriage at 2014-07-15 will create a new row in the dimension table with the ValidFrom 2014-07-15 because she took her husband's surname to Marie Sainte, and the previous row will get ValidTo 2014-07-15 meaning that until that date she kept her first married name.

Marie's wedding dates is in general called a business date or a date of change to determine a new value in a ValidFrom column, read more in section 6.5.2.

After Marie changes her name again in a Customer dimension type 2.

Customer_key	**CustomerId_bkey**	**CustomerName**	**ValidFrom**	**ValidTo**
-2	-2	Unknown	1900-01-01	9999-12-31
-1	-1	Missing	1900-01-01	9999-12-31
1	176	Hans Andersen	1900-01-01	9999-12-31
2	359	Joe Watson	1900-01-01	9999-12-31
3	421	Marie Beaulieur	1900-01-01	2007-11-16
4	421	Marie Lesauvage	2007-11-16	2014-07-15
5	421	Marie Sainte	2014-07-15	9999-12-31

Business key CustomerId_bkey is the »glue« that holds the multiple rows together.

Timeline metadata columns ValidFrom and ValidTo have information about the data row itself. A dimension can have extra columns for metadata for a data row in a table e.g.: IsCurrent, IsDeleted, IsInferred, State column or Status column with values: »Current, Deleted, Expired«. More about metadata columns in section 6.3.

Marie has made six purchases and they have been typed into a source system table together with CusomerId and PurchaseDate. When we transfer the six purchases and let them entered into a fact table, the combination of CusomerId and PurchaseDate will lookup in the Customer dimension to fetch a dimension key value in column Customer_key to a fact row by a rule in a sql statement that reminds of PurchaseDate Between ValidFrom And ValidTo with match of the business key CustomerId_bkey:

```
CustomerId_bkey = CustomerId AND
PurchaseDate >= ValidFrom AND PurchaseDate < ValidTo
```

Another formulation of the rule in a sql statement:

```
CustomerId_bkey = CustomerId AND
ValidFrom <= PurchaseDate AND ValidTo > PurchaseDate
```

The rule has to find one and only one row in the Customer dimension of type 2, where business key in column CustomerId_bkey match with a source system primary key column CustomerId, and where the date of purchase has to be between a range of the dimension columns ValidFrom and ValidTo according to the first rule, and according to the second rule that ValidFrom is before a PurchaseDate and a ValidTo is after a PurchaseDate.

It is called **Date Range Lookup** to fetch a dimension key value in a dimension table through a business key value and a transaction date value, and save a dimension key value in a fact table row. (In danish dato-spænd opslag)

An example for a business key column CustomerId with a value 421 and a transaction date column PurchaseDate with value 2014-05-08 for a date range lookup in a Customer dimension to fetch a dimension key value in column Customer_key to a fact row with this sql execution:

```
CustomerId_bkey = 421 AND
2014-05-08 >= ValidFrom AND 2014-05-08 < ValidTo
```

The dimension row with ValidFrom 2007-11-16 and ValidTo 2014-07-15 meets the criteria for this sql execution:

```
CustomerId_bkey = 421 AND
2014-05-08 >= 2007-11-16 AND 2014-05-08 < 2014-07-15
```

The dimension row has dimension key column Customer_Key value 4, which will be used in the inserted fact table row to represent Marie's purchase at 2014-05-08, where her surname was Lesauvage at that time. The fact table row refers to the dimension key value to fetch the registered values of the dimension at the time when fact data was happening, occurred, registered or took place.

Date Range Lookup will be performed for each row of purchase from a source system, where each business key CustomerId and PurchaseDate must meet the above criteria for dimension Customer to fetch dimension key Customer_key into a fact table row.

If a business key CustomerId has no value as null, then dimension key Customer_key gets default value -1 for Missing into a fact table row.

If a business key CustomerId has a value that is not found and not exists in dimension Customer dimension in dimension key Customer_key, then dimension key Customer_key gets default value -2 for Unknown into a fact table row.

An example of a fact table where Marie has made many purchases since year 2005 and each purchase has its own Customer_Key value as a navigation reference back to the Customer dimension to fetch full name when a purchase was happening, occurred, registered or took place.

Sales fact with dimension key columns to dimensions of type 2.

PurchaseDate_key	Customer_key	Product_key	Quantity	SaleAmount
2005-11-10	1	21	1	45
2005-11-10	2	29	2	140
2005-12-01	3	32	2	200
2010-06-30	4	17	1	65
2014-05-08	4	32	3	300
2014-08-16	5	21	2	90
2016-03-15	5	29	1	70
2016-03-15	5	42	1	100
2016-03-31	-1	45	1	0
2016-04-01	-2	-2	2	250

For each row in the fact table, a dimension key value in column Customer_key refers to a row in a Customer dimension table to present a name of the customer at the time a purchase was happening, occurred, registered or took place. Customer_key column in fact table represents **as-was** when it is joined to the Customer dimension table.

At PurchaseDate 2016-03-31 there is a missing customer with key value -1 because a product has been stolen and no money into the cash register.

At PurchaseDate 2016-04-01 there is an unknown customer with key value -2 because a IT software upgrade failed to register a new customer who purchased a new product. Therefore both business keys is not known in the dimensions.

An example of a SQL statement to enter data from a source system table called SAP_Purchasing into a fact table called FactSales using **type 2 dimensions** by date range lookups for customer and product. Key value -2 handle unknown if a business key value does not exists in a dimension. Key value -1 handle missing if a business key value is null from a source system because it was missing there e.g. a unwanted sale because a product has been stolen therefore the customer business key value is null.

The data capture is incremental load for column PurchasingId to fetch the greatest value from FactSales table to fetch the archive table rows with values greater than it.

Database schema dma represents a data mart area and schema ara represents an archive area where data from a source system SAP_Purchasing is stored.

To avoid a null in a dimension key in a fact table (called »handling null foreign keys in fact tables«) two integer artifact values or inferred members are used: -1 refers to missing and -2 refers to unknown.

For example in a sql statement with three dimension key columns for Date, Customer and Product with suffix _key, and two business key columns with suffix _bkey:

```
INSERT INTO dma.FactSales
  (PurchaseDate_key, Customer_key, Product_key, Quantity, SaleAmount, PurchasingId)
SELECT pur.PurchaseDate,
      Customer_key = ISNULL(cus.Customer_key,-2), -- in case of no match in join.
      Product_key = ISNULL(pro.Product_key,-2),   -- in case of no match in join.
      pur.Quantity,
      pur.SaleAmount,
      pur.PurchasingId
FROM archive.SAP_Purchasing pur
    LEFT OUTER JOIN dma.DimCustomer cus
      ON cus.CustomerId_bkey = ISNULL(pur.CustomerId,-1) AND
        pur.PurchaseDate >= cus.ValidFrom AND pur.PurchaseDate < cus.ValidTo
    LEFT OUTER JOIN dma.DimProduct pro
      ON pro.ProductNumber_bkey = ISNULL(pur.ProductNumber,-1) AND
        pur.PurchaseDate >= pro.ValidFrom AND pur.PurchaseDate < pro.ValidTo
WHERE pur.PurchasingId > (SELECT MAX(PurchasingId) FROM dma.FactSales)
```

A ETL process is represented by the three operations in the SQL statement:

- Extracting: Select From Where for incremental load from an archive area.
- Transforming: Left Outer Join IsNull to lookup the dimension _key values.
- Loading: Insert Into a fact table in the data mart area.

The use of Left Outer Join is for the unfortunate situation, where a dimension table is not up-to-date with all data. If a business key value from a source data table is not known in the data warehouse and therefore does not exists in a dimension table, then the entered fact table row will get a key value -2 in the fact column of the dimension as a reference to the unknown row in the dimension. If the business key has null then the entered fact table row will get a key value -1 in the fact column of the dimension as a reference to the missing row in the dimension, so the Join part does handle a purchase is without a customer or is without a product.

The above sql statement do a **batch bulk insert of data into a fact table**. An alternative to key value -2 for unknown is an inferred member as described in section 4.5 which is done with a **streaming insert of data into a fact table** with a ETL process implemented through a tool e.g. SSIS, because then we can handle the first time an unknown appears by adding it to the dimension and receive a new key value, and when the unknown reappears later in the same data stream, it is already in place in the dimension with a known key value.

Customer current names for a type 2 dimension
When a report of sales wants to show the customers current name (most recent name) independent of the date of purchase, it is solved with a database view that will show this **non-unique list** of most recent CustomerName for each key value of column Customer_key:

Customer_key	CustomerId	CustomerName
-2	-2	Unknown
-1	-1	Missing
1	176	Hans Andersen
2	359	Joe Watson
3	421	Marie Sainte
4	421	Marie Sainte
5	421	Marie Sainte

I recommend using SQL views upon a type 2 dimension for a current view and a registered view to present the wanted dimension values from a fact table, like FactSales, or from a view upon a fact table.

Dimension key columns in a fact table e.g. Customer_key and Product_key does not tell whether the dimension is a type 1 or a type 2, but when I join from a fact table dimension key, e.g. Customer_key to a dimension view called either DimCustomer_Current or DimCustomer_Registered
I choose which dimension values I want to fetch and see.

Lets see how to implement these views for current values and for registered values.

A **type 2 current view dimension** implemented using ValidTo and a self join:

```
CREATE VIEW DimCustomer_Current AS
SELECT d.Customer_key, d.CustomerId_bkey AS CustomerId, c.CustomerName
FROM DimCustomer d
      INNER JOIN  -- self join for a type 2 dimension to fetch current values.
        (SELECT CustomerId_bkey, CustomerName  -- the most recent CustomerName.
         FROM DimCustomer
         WHERE ValidTo = '9999-12-31'  -- datetime2(7) '9999-12-31 23:59:59.9999999'
        ) c ON c.CustomerId_bkey = d.CustomerId_bkey
```

There can be an index on ValidTo + CustomerId_bkey in case of performance issue.

Implemented using IsCurrent and a self join:

```
CREATE VIEW DimCustomer_Current AS
SELECT d.Customer_key, d.CustomerId_bkey AS CustomerId, c.CustomerName
FROM DimCustomer d
      INNER JOIN  -- self join for a type 2 dimension to fetch current values.
        (SELECT CustomerId_bkey, CustomerName  -- the most recent CustomerName.
         FROM DimCustomer
         WHERE IsCurrent = 1  -- True
        ) c ON c.CustomerId_bkey = d.CustomerId_bkey
```

The view will do a query to select the current values for each key value in a dimension to a non-unique list and show a result set, rowset or recordset as showned in the table above.

If your sql has windowing function the view can avoid the self join which can enhance performance and here is no use of ValidTo column or IsCurrent column:

```
CREATE VIEW DimCustomer_Current AS
SELECT Customer_key, CustomerName = LAST_VALUE(CustomerName)
          OVER(PARTITION BY CustomerId_bkey ORDER BY ValidFrom
          ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING)
FROM DimCustomer
```

Use view DimCustomer_Current in a join with a Sales fact table to fetch and see **customer current values for each fact row**:

```
SELECT d.CustomerName, f.PurchaseDate_key, f.Quantity, f.SaleAmount
FROM FactSales f
      INNER JOIN DimCustomer_Current d ON d.Customer_key = f.Customer_key
```

Customer_key column from the view DimCustomer_Current represent **as-is** for most recent data when it is joined to the fact table.

The disadvance of the second view for current values of the dimension is that the windowing function must be used for each column that is wanted to be showed from the dimension table.

The disadvance of the first view for current values of the dimension is the join on the business key because it can be a string representation or a composite business key is composed of multiple columns that needs to be part of the Inner Join columns, which can cost performance when the dimension has many rows.

A key mapping table can transform a business key value to extra columns as:

- immutable surrogate business key (suffix _sbkey) e.g. a code which can be understood by a human
- artificial auto-generated unique sequence number or an identity column as a durable key (suffix _dkey) as an integer representation of a business key.

Adding a durable key column to a dimension (and adding a surrogate business column for easy search criteria) we get an **extension of type 2** as a look-alike type 7 dimension, where we let the durable key be used in a view DimCustomer_Current for faster performance:

```
CREATE VIEW DimCustomer_Current AS
SELECT d.Customer_key, d.CustomerId_bkey AS CustomerId, c.CustomerName
FROM DimCustomer d
       INNER JOIN  -- self join for a type 2 dimension to fetch current values.
         (SELECT Customer_dkey, CustomerName  -- the most recent CustomerName.
          FROM DimCustomer
          WHERE ValidTo = '9999-12-31'  -- datetime2(7) '9999-12-31 23:59:59.9999999'
         ) c ON c.Customer_dkey = d.Customer_dkey
```

When performance of join a fact table to a _Current view is not fast enough, we can rename a view to _Current_Build as a materialized view, where we let a ETL process insert current values into a table e.g. StagingDimCustomer_Current and make a simple view DimCustomer_Current:

```
CREATE VIEW DimCustomer_Current AS
SELECT Customer_key, CustomerId, CustomerName
FROM StagingDimCustomer_Current
```

and use the view in a join to a fact table.

A **type 2 registered view dimension** can be implemented like this:

```
CREATE VIEW DimCustomer_Registered AS
SELECT Customer_key, CustomerId_bkey AS CustomerId, CustomerName
FROM DimCustomer
```

Use view DimCustomer_Registered in a join with a Sales fact table to fetch and see **registered customer value for each fact row**:

```
SELECT d.CustomerName, f.PurchaseDate_key, f.Quantity, f.SaleAmount
FROM FactSales f
       INNER JOIN DimCustomer_Registered d ON d.Customer_key = f.Customer_key
```

Customer_key column from the view DimCustomer_Registered represent **as-was** when fact data was happening, occurred, registered or took place or was entered into the fact table.

In case you want to fetch and see the **original value** of the dimension as type 0:

```
CREATE VIEW DimCustomer_Original AS
SELECT Customer_key, CustomerName = FIRST_VALUE(CustomerName)
          OVER(PARTITION BY CustomerId_bkey ORDER BY ValidFrom
          ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING)
FROM DimCustomer
```

No ValidTo column – how to calculate or compute ValidTo (or EndDate)
There is another type 2 approach that does not have a ValidTo column in a dimension table, and therefore a ETL process does not have to update the previous row. Instead a ValidTo column will be calculated or computed when it is needed to determine a dimension key for a forthcoming fact row:

```
CREATE VIEW DimCustomer_ValidTo AS
SELECT Customer_key, CustomerName, ValidFrom,
         ValidTo = LEAD(ValidFrom, 1, '9999-12-31')
                   OVER(PARTITION BY CustomerId_bkey ORDER BY ValidFrom)
FROM DimCustomer
```

In mysql 5.7:

```
CREATE VIEW DimCustomer_ValidTo AS
SELECT Customer_key, CustomerName, ValidFrom,
  (SELECT IFNULL(MIN(b.ValidFrom), '9999-12-31 23:59:59') AS ValidFrom
   FROM DimCustomer b
   WHERE b.CustomerId_bkey = a.CustomerId_bkey AND
            b.Customer_key > a.Customer_key
  ) AS ValidTo
FROM DimCustomer a
```

Customer current names for a dropdown box search criteria
When a data access BI tool like Excel, Power BI, Tableau, QlikView, Qlik Sense, Alteryx or Targit wants to show the customers current name (most recent name) independent of the date of purchase, it is solved with a database view that will show this **unique list** of most recent CustomerName for each key value of column Customer_Current_key:

Customer_Current_key	CustomerName
-2	Unknown
-1	Missing
1	Hans Andersen

2	Joe Watson
5	Marie Sainte

```
CREATE VIEW DimCustomer_Current AS
SELECT Customer_key AS Customer_Current_key, CustomerName
FROM DimCustomer
WHERE ValidTo = '9999-12-31'  -- datetime2(7) '9999-12-31 23:59:59.9999999'
```

A business user can in a dropdown box select Marie Sainte (meaning Customer_key value 5) and all her six purchases facts will be shown or a sum of SaleAmount even though she has changed her name over time.

A FactSales_Current view below will include a Customer_Current_key column to be joined to the above view DimCustomer_Current in a BI tool's data model.

PurchaseDate_key	Customer_key	Customer_Current_key	Product_key	Quantity	Sale Amount
2005-11-10	1	1	21	1	45
2005-11-10	2	2	29	2	140
2005-12-01	3	5	32	2	200
2010-06-30	4	5	17	1	65
2014-05-08	4	5	32	3	300
2014-08-16	5	5	21	2	90
2016-03-15	5	5	29	1	70
2016-03-15	5	5	42	1	100
2016-03-31	-1	-1	45	1	0
2016-04-01	-2	-2	-2	2	250

```
CREATE VIEW FactSales_Current AS
WITH dimCustomer_Current (Customer_key, Customer_Current_key) AS
(
SELECT Customer_key,
      MAX(Customer_key) OVER(PARTITION BY CustomerId_bkey) AS Customer_Current_key
FROM DimCustomer
)
SELECT f.PurchaseDate_key, f.Customer_key, d.Customer_Current_key, f.Product_key,
      f.Quantity, f.SaleAmount
FROM FactSales f
       INNER JOIN dimCustomer_Current d ON d.Customer_key = f.Customer_key
```

When current customer Marie Sainte is selected in a dropdown box, a BI tool will do a query based on the BI tool's data model to show all her six purchases facts even though she has changed her name over time:

```
SELECT d.CustomerName, f.PurchaseDate_key, f.Quantity, f.SaleAmount
FROM FactSales_Current f
       INNER JOIN DimCustomer_Current d
         ON d.Customer_Current_key = f.Customer_Current_key
WHERE d.CustomerName = 'Marie Sainte'
```

CustomerName	PurchaseDate_key	Quantity	SaleAmount
Marie Sainte	2005-12-01	2	200
Marie Sainte	2010-06-30	1	65
Marie Sainte	2014-05-08	3	300
Marie Sainte	2014-08-16	2	90
Marie Sainte	2016-03-15	1	70
Marie Sainte	2016-03-15	1	100

The fact view FactSales_Current is still keeping the Customer_key column to be joined to the Customer dimension for showing the name of a customer when a purchase was happening, occurred, registered or took place.

The fact view FactSales_Current is using a With part (Common Table Expression) to make a temporary result set. It could be changed to a materialized view which store data in a table with a clustered primary key through a ETL process, and table FactSales will join to the materialized-view-table to get a better performance.

6.2.3. Type 7 dimension

When a dimension is a type 7 it keeps history as type 2 via **dimension key**, and a type 7 does the same as type 1 current value with an extra column called **durable key** as an integer representation of a business key, because a business key value can be a mix of number and text or can be a composite business key. A durable key will be generated in a **key mapping table** and fetched by a business key.

Customer dimension is a type 7 with columns for a dimension key _key, a durable key _dkey and a business key _bkey.

Customer	Customer	Customer	CustomerName	ValidFrom	ValidTo

_key	_dkey	Id_bkey			
-2	-2	-2	Unknown	1900-01-01	9999-12-31
-1	-1	-1	Missing	1900-01-01	9999-12-31
1	1	176	Hans Andersen	1900-01-01	9999-12-31
2	2	359	Joe Watson	1900-01-01	9999-12-31
3	3	421	Marie Beaulieur	1900-01-01	2007-11-16
4	3	421	Marie Lesauvage	2007-11-16	2014-07-15
5	3	421	Marie Sainte	2014-07-15	9999-12-31

The dimension key as primary key in column Customer_key is holding the type 2 complete history of the dimension columns changes.

The durable key in column Customer_dkey is holding the type 1 current version of the dimension columns.

Both dimension key Customer_key and durable key Customer_dkey will appear in a fact table as a dual foreign key for a given dimension.

Sales fact with dimension keys and durable keys columns to dimensions of type 7.

Purchase-Date_key	Customer_key	Customer_dkey	Product_key	Product_dkey	Quantity	Sale Amount
2005-11-10	1	1	21	11	1	45
2005-11-10	2	2	29	14	2	140
2005-12-01	3	3	32	19	2	200
2010-06-30	4	3	17	17	1	65
2014-05-08	4	3	32	19	3	300
2014-08-16	5	3	21	11	2	90
2016-03-15	5	3	29	14	1	70
2016-03-15	5	3	42	19	1	100
2016-03-31	-1	-1	45	11	1	0
2016-04-01	-2	-2	-2	-2	2	250

For each row in the fact table, the key value in column Customer_key is pointing to a row in the dimension table and give the name of the customer at the time a purchase was happening, occurred, registered or took place. Customer_key column in fact table represents **as-was** when it is joined to the dimension table.

For each row in the fact table, the durable key value in column Customer_dkey is pointing to a row in a current view of the dimension table to give the current name and most recent name of the customer. Customer_dkey column in fact table represents **as-is** when it is joined to the dimension table with ValidTo = 9999-12-31.

The durable key in column Customer_dkey will appear in a current view of the dimension table that will show each customer in a **unique list** of most recent customers.

Customer_dkey	CustomerId	CustomerName
-2	-2	Unknown
-1	-1	Missing
1	176	Hans Andersen
2	359	Joe Watson
3	421	Marie Sainte

A **type 7 current unique view dimension** can be implemented like this:

```
CREATE VIEW DimCustomer_Current_Unique AS
SELECT Customer_dkey, CustomerId_bkey AS CustomerId, CustomerName
FROM DimCustomer
WHERE ValidTo = '9999-12-31'  -- datetime2(7) '9999-12-31 23:59:59.9999999'
```

Implemented using IsCurrent:

```
CREATE VIEW DimCustomer_Current_Unique AS
SELECT Customer_dkey, CustomerId_bkey AS CustomerId, CustomerName
FROM DimCustomer
WHERE IsCurrent = 1  -- True
```

Durable key _dkey column will appear in a join with a fact table to fetch and see **current values for each fact row**:

```
SELECT d.CustomerName, f.PurchaseDate_key, f.Quantity, f.SaleAmount
FROM FactSales f
      INNER JOIN DimCustomer_Current_Unique d
        ON d.Customer_dkey = f.Customer_dkey
```

In a data access BI tool like Excel, Power BI, Tableau, QlikView, Qlik Sense, Alteryx or Targit the type 7 current view becomes a dimension of most recent values and inside the BI tool's data model the view is joined to the fact table's durable key co-

lumn. A business user can in a dropdown box select Marie Sainte (meaning Customer_dkey value 3) and all her six purchases facts will be shown or a sum of SaleAmount even though she has changed her name over time.

Did you notice, that Product_key value 42 is a new version of a product because it has Product_dkey value 19 and since Product_key value 32 also has Product_dkey value 19, they are all of the same product in different versions which could be the category of the product that has changed over time.

The advance with type 7 is that a business key from a source system is not part of a query to fetch and see the most recent, current values from a dimension. Read about SCD type 7 dimension in next section.

6.3. Kimball type 7 dimension alround with examples

All about type 7 dimension including suffixes for columns, audit columns, metadata columns and examples.

6.3.1. Type 7 dimension

A type 7 dimension will keep all values in a dimension independent of a source system with an option to easy to roll up historical facts based on current dimension characteristics. It is formerly known as »Compliance-Enabled Data Warehouses« in Kimball Design Tip #74 that achieve:

- Eliminating Type 1 and Type 3 overwrites.
- Accessing all prior versions of a database at points in time.
- Protecting the custody of your data.
- Showing why and when changes to data occurred.

Durable key was presented in Kimball Design Tip #147 Durable "Super-Natural" Keys, and type 7 in Kimball Design Tip #152 Slowly Changing Dimension Types 0, 4, 5, 6 and 7.

With type 7 dimension Kimball introduced an integer representation of a business key and he called it **durable key**, and it will be implemented as an artificial auto-generated unique sequence number as a surrogate key value for each business key value in a key mapping table.

Durable means there is no business rule that can change the key value because it belong to the data warehouse. A business key CustomerId with value 421 gets a durable key identifier with value 3, and it will be used in fact table to referring to the customer. When the CRM system will be replaced with a new system in the cloud, it is a task for the data warehouse to map the old CustomerId (421) to the new source system CustomerId (836) and thereby keep the durable key value 3 so the old fact rows will be unchanged. If a CustomerNumber has same value in both systems, it will be very helpful for the mapping and integration. That makes the data warehouse resilient and robust.

Kimball said: »Slowly changing dimension type 7 is the final hybrid technique used to support both as-was and as-is reporting. A fact table can be accessed through a dimension modeled both as a type 1 dimension showing only the most current values, or as a type 2 dimension showing correct contemporary historical profiles. The same dimension table enables both perspectives.« source.

Both the durable key and the dimension key is stored in a fact table.

For the type 1 perspective, the current flag in a dimension is constrained to be current, and a fact table is joined via the durable key to fetch the current values for the fact row.

For the type 2 perspective a fact table is joined via the dimension key to fetch the registered values for the fact row.

The two perspectives will be deployed as separate views to a BI tool in the Presentation interface area (PIA).

In this section, I like to give my own description of a dimension type 7 together with an example and how to join data from fact to dimensions.

I will start with life cycle terms for a value, a member or data in a dimension:

- **Original** is the value of birth when it entered and occurred first time.
- **Registered** is the value when data happening, occurred or took place.
- **Historical** is tracking the value for changes (modified/updated) at any time.
- **Current** is the value of the most recent data, active or actual.
- **Deleted** is the value no longer available in a source system.

Using of four idioms to set a date and time on a value from a dimension:

- Registered stands for **as-was** when data happening, occurred or took place.
- Current stands for **as-is** for most recent data.
- Original stands for **as-began** when data was opened, appeared or borned.
- Historical stands for **as-of** for past data at a specific reporting date or at any arbitrary date in the past as a point-in-time analysis.

An additional information for a dimension value (dimension data) would be that it is not longer available in a source system because it has been marked expired or purged or it has been deleted **as-deleted**.

When I have one fact data row in my hand, I can provide **five versions** of the values in a dimension, therefore I like to call them five perspectives:

- **Registered** is the dimension value when the fact data row was registered or entered into the data warehouse with a transaction date or an insert time.
- **Current** is the most recent dimension value in a non-unique list and thus independent of the time of registration of the fact data row.
- **Current** is the most recent dimension value in a unique list and thus independent of the time of registration of the fact data row.
- **Original** is the entered first time dimension value maybe even before the fact data row exists in a source system.
- **Historical** is the dimension value at a specific given reporting date point-in-time independent of the fact transaction date.

6.3.2. Design pattern for a type 7 dimension with extra key columns

Different suffixes for columns.

_key stands for a **dimension key** is an artificial auto-generated unique sequence number or an identity column as a surrogate key for the primary key in a dimension table, normally starting from 1 with a new integer surrogate number every time a value in a source system is changed and send to the data warehouse. The dimension key _key will be in a fact table to join on a dimension to fetch the registered values. It is a reference from a fact table row to a dimension table row with descriptive data columns at the time the fact data row was registered or entered into the data warehouse. Dimension key _key is used for lookup in dimension for the **registered values** and **current values** in a non-unique list.

_dkey stands for a **durable key** which is not a unique integer number per row, instead it will hold the same number as the dimension key _key column when a dimension value entered the dimension table for the first time and the number will be repeated every time the value change. A durable key is an integer representation of a **business key** and it will be generated in a key mapping table and fetched by a business key value. The durable key _dkey will be in a fact table to join on a dimension to fetch the current values, the original values or the historical values at a specific given reporting date point-in-time. Durable key _dkey is used for lookup in dimension for the **current values** in a unique list, for the **original values** when it was entered the first time or for the **historical values** at a specific given reporting date point-in-time.

_pkey stands for a **previous dimension key** (or parent-child key) which is a reference to a dimension key value for the previous version of a dimension value when a value change over time. It is a referential integrity where _pkey is a foreign key for the dimension key with one exception that value 0 in _pkey does not has a reference to a _key value because value 0 represent the first time the dimension value enter a dimension table and therefore has no previous value and that is the same as the original value.

_bkey stands for a **business key** which is unique in a source system like a primary key. A composite business key is composed of multiple columns like Name_bkey + Street_bkey + Zipcode_bkey and they will be multiple columns in the dimension table. Sometimes a string concatenation with a separator character to handle them in one _bkey column to compare with source data. Identical data can exists in multiple source systems like an employee in CRM, ERP and HR with their own business key in the Employee dimension through sibling columns like CRM_bkey, ERP_bkey and HR_bkey with different representations like a number, a code and an email address. Depending of complexity there is a need of a key mapping table in a data warehouse to take care of the merge of business keys and sometimes it is a manual task to merge data together when there is no suitable natural key. Importance is to have an employee as one entity with one durable surrogate number for the combination of business keys making a one-to-one relationship between the _bkey and the _dkey. Examples of concatenated columns to a checksum column called:

```
Comparison_bkey_meta =
  convert(binary(32),hashbytes('sha2_256',
    concat(upper(rtrim(ltrim(Name_bkey))),';',
           upper(rtrim(ltrim(Street_bkey))),';',Zipcode_bkey))
```

The other descriptive data columns to a computed column called:

```
Comparison_data_meta AS
```

```
convert(binary(32),hashbytes('sha2_256',
  concat_ws(';',upper(trim(FirstName)),upper(trim(LastName)),
  convert(varchar(23),BirthDate,(126))))) Persisted
```

used to find out when descriptive data values has been changed in a source system to make a new row in dimension with new validfrom/validto in a ETL process.

_skey stands for a **surrogate key** which is a primary key in a source system as an artificial auto-generated unique sequence number or an identity column (id, uid). Helpful to create a conformed dimension across multiple source systems which have their own business key that can't be used in the merge because it is a surrogate key identity column as a unique sequence number. Also helpful when a business key can change its value in a source system and a dimension needs to be updated.

_sbkey stands for a **surrogate business key** which is a unique text value per row that is immutable (unchangeable) to provide a code to be easy-to-read for a business user and a developer. It will be generated in a key mapping table. I am using a Code column when a dimension contains its own data members to for an enrichment and is not loaded from a source system. Therefore a column for _sbkey and _bkey is seldome in the same dimension. Often I naming _sbkey as **Code**. Example to fetch »End of Support« from a dimension by using a Code column:

```
INNER JOIN dim.Lifecycle dl ON dl.LifecycleCode = 'EOS'.
```

Using the _key number value has a big risk of another surrogate number value between server environments for Development, Test and Production. If the text should change from »End of Support« to »End of Service Life«, I will keep the code value 'EOS', therefore I don't need to change my programming. Instead of _sbkey or Code I have sometimes used **_ckey** for code key.

If you not like using the underscore or CamelCase, another notation is usable e.g.:
ClientKey, ClientDkey, ClientPkey, ClientBkey, ClientSkey,
SoftwareRoleKey, SoftwareRoleCode, SoftwareRoleName, SoftwareRoleDescription, SoftwareRoleOrder, ComparisonBkeyMetadata, ComparisonDataMetadata.

6.3.3. Example of a type 7 dimension table and a fact table

Let's have an example where a source system has changed its value three times from »Warning« to »Notification« and to »Caution«. Each of the three values become a new row in a **dimension table** together with two metadata columns that representing the period for which the value was found in a source system, here called:

- **ValidFrom** also called EffectiveDate or StartDateTime or BeginDateTime.
- **ValidTo** also called ExpirationDate or StopDateTime or EndDateTime.

A System dimension type 7 with key, metadata and descriptive data columns.

System _key	System _dkey	System _pkey	System _bkey	System	ValidFrom	ValidTo
19	19	0	76CB	Warning	1900-01-01 00:00:00	2015-05-08 14:12:21
35	19	19	76CB	Notification	2015-05-08 14:12:21	2015-07-31 08:45:48
87	19	35	76CB	Caution	2015-07-31 08:45:48	9999-12-31 23:59:59

_key stands for a dimension key as a unique surrogate sequential number.
_dkey stands for a durable key as an integer representation of a business key and comes from a key mapping table.
_pkey stands for a previous key as a reference to a _key value for the previous.
_bkey stands for a business key as mostly the primary key in a source data.

Dimension key in column _key is a unique primary key of the dimension table.
The combination of columns _dkey + ValidFrom is unique too.
The combination of columns _bkey + ValidFrom is unique too.

»Warning« and »Notification« is history values of the System dimension, and »Caution« is the most recent, current value of the System dimension with these labels:

- »Warning« is the original value.
- »Notification« is the historical value.
- »Caution« is the current value.
- »?« is a registered value depending of a transaction date in a fact row.

The challenge is that you have to choose which version of a dimension value you like to present in your report.

Registered value
From a source system the data warehouse has received two outages fact data with an OutageDate (happening time or transaction date) and a SystemId business key

that match to System_bkey in the System dimension. Example of two source data rows.

OutageDate	SystemId	ProductionKWh
2015-05-25 10:22:43	76CB	173
2015-08-02 15:47:10	76CB	221

For each source data row the value of OutageDate will carry out a date range lookup in the System dimension at the time when an outage was happening to fetch a dimension key value in column System_key and a durable key value in column System_dkey. The fact data row is registered with criteria e.g.:

```
System_bkey = SystemId AND
OutageDate >= ValidFrom AND OutageDate < ValidTo
```

or another formulation of the criteria e.g.:

```
System_bkey = SystemId AND
ValidFrom <= OutageDate AND ValidTo > OutageDate
```

It will select one row from the dimension table and the values of columns System_key and System_dkey will be copied to the new fact row, like for these two outages incidents (in danish episode, tilfælde, noget der sker tilfældigt) at Maj 25 and August 2 in 2015 will become two fact data rows.

OutageDate	System_key	System_dkey	ProductionKWh
2015-05-25 10:22:43	35	19	173
2015-08-02 15:47:10	87	19	221

System_key is a reference to the System dimension value at the registered fact time from column OutageDate, and System_dkey is a reference to the entity with business key 76CB to provide current, original or historical value.

When we like to see the value from the dimension at the time where an outage was happening, the fact table column System_key will join to the dimension key column System_key and get the value from the unique row. I call this value the registered value, because it was the value of the dimension when an outage incidents was happening, occurred, registered or took place, and the value will never change because the dimension is handling all changes of values.

Fact table join to the dimension table to fetch and see registered value using the dimension key:

```
SELECT d.System, f.OutageDate, f.ProductionKWh
FROM   fact.Outage f
       INNER JOIN dim.System d ON d.System_key = f.System_key
```

System	OutageDate	ProductionKWh
Notification	2015-05-25 10:22:43	173
Caution	2015-08-02 15:47:10	221

Here we see the purpose of the OutageDate and in case of the source system does not provide a »transaction date« to fetch a dimension key value, the data warehouse will use an InsertTime (receipt time, registered time, recording time, current_timestamp, getdate, sysdatetime, created date) for when the fact data is inserted, registered or entered into the fact table row with criteria:

```
dim.System_bkey = fact.SystemId AND
fact.InsertTime >= dim.ValidFrom AND fact.InsertTime < dim.ValidTo
```

Current value
It can sometimes confuse business users because the value of System has changed over time, and what is the summation of ProductionKWh of the System when using the registered value? Therefore we can provide the current value of the System dimension.

Fact table join to the dimension table to fetch and see current value using the durable key:

```
SELECT d.System, f.OutageDate, f.ProductionKWh
FROM   fact.Outage f
       INNER JOIN dim.System d
         ON d.System_dkey = f.System_dkey AND
            d.ValidTo = '9999-12-31 23:59:59'
```

System	OutageDate	ProductionKWh
Caution	2015-05-25 10:22:43	173
Caution	2015-08-02 15:47:10	221

And a summation of ProductionKWh is 394 KWh for Caution.

Original value
We can provide an original value of the System dimension by using the durable key.

Fact table join to the dimension table to fetch and see original value:

```
SELECT d.System, f.OutageDate, f.ProductionKWh
FROM   fact.Outage f
       INNER JOIN dim.System d
         ON d.System_dkey = f.System_dkey AND
            d.System_pkey = 0
```

System	OutageDate	ProductionKWh
Warning	2015-05-25 10:22:43	173
Warning	2015-08-02 15:47:10	221

And a summation of ProductionKWh is 394 KWh for Warming, which is the original value of the system, maybe some business users remember it best.

Historical value
We can provide a historical value of the System dimension at a specific given reporting date point-in-time (ReportTime, RequestedDate) that will be part of the criteria by using the durable key.
Fact table join to the dimension table to fetch and see historical value:

```
DECLARE @ReportTime datetime = '2015-06-01 10:24'

SELECT d.System, f.OutageDate, f.ProductionKWh
FROM   fact.Outage f
       INNER JOIN dim.System d
         ON d.System_dkey = f.System_dkey AND
            @ReportTime >= d.ValidFrom AND @ReportTime < d.ValidTo
```

System	OutageDate	ProductionKWh
Notification	2015-05-25 10:22:43	173
Notification	2015-08-02 15:47:10	221

The »between-filter« do the constrained to a single version value of system. A reporting datetimestamp at »June 1, 2015, 10.24« will fetch system Notification and that will never change even though the system name changes value over time. It means I can always repeat and reprocedure my report as long as I remember the reporting datetimestamp.

If I choose another reporting datetimestamp like »September 1, 2015, 09.46« it will fetch system Caution. The reality with this approach is that you have a many-to-many relationship between the fact table and the dimension, so you must force your delivery reporting tools to choose an as-of date to allowing point-in-time analysis at a given date to avoid bringing back all combinations and resulting in overstated measurements.

Helper views
It is hard to remember the criteria for _key, _dkey, ValidFrom and ValidTo therefore I will make some sql helper views to do a join between dimension and fact more easy for business users and analysis tools.

Registered view with dimension key

```
CREATE VIEW dim.System_Registered AS
SELECT System_key, System
FROM   dim.System
```

Used in join with fact table to fetch and see registered value for a fact row:

```
SELECT d.System, f.OutageDate, f.ProductionKWh
FROM   fact.Outage f
       INNER JOIN dim.System_Registered d ON d.System_key = f.System_key
```

System	OutageDate	ProductionKWh
Notification	2015-05-25 10:22:43	173
Caution	2015-08-02 15:47:10	221

When an Outage was happening.

Current view with dimension key

```
CREATE VIEW dim.System_Current AS
SELECT d.System_key, c.System
```

```
FROM   dim.System d
       INNER JOIN  -- self join for a type 7 dimension to fetch current values.
        (SELECT System_dkey, System  -- the most recent System.
         FROM dim.System
         WHERE ValidTo = '9999-12-31 23:59:59'
        ) c ON c.System_dkey = d.System_dkey
```

The view will do a query to select the current values for each key values in a dimension to a non-unique list, because the current values will use the dimension key _key column.

Used in join with fact table to fetch and see current value for a fact row:

```
SELECT d.System, f.OutageDate, f.ProductionKWh
FROM   fact.Outage f
       INNER JOIN dim.System_Current d ON d.System_key = f.System_key
```

System	OutageDate	ProductionKWh
Caution	2015-05-25 10:22:43	173
Caution	2015-08-02 15:47:10	221

When an Outage is current.

Let us imagine we are back in time at June 5, 2015 and we like to see in a BI tool the current system of the outage at May 25, the view System_Current will provide system Notification because that was the current value at that time.

System	OutageDate	ProductionKWh
Notification	2015-05-25 10:22:43	173

When we do the same at August 22, the view System_Current will provide system Caution.

Current unique view with durable key and no self join

```
CREATE VIEW dim.System_Current_Unique AS
SELECT System_dkey, System
FROM   dim.System
WHERE ValidTo = '9999-12-31 23:59:59'
```

The view will do a query to select the current values for each key values in a dimension to a unique list, because the current values will use the durable key _dkey column. The unique list of current values can be used in a BI tool for a dropdown box search criteria for a report and for a dashboard.

Used in join with fact table to fetch and see current value for a fact row:

```
SELECT d.System, f.OutageDate, f.ProductionKWh
FROM   fact.Outage f
       INNER JOIN dim.System_Current_Unique d ON d.System_dkey = f.System_dkey
```

System	OutageDate	ProductionKWh
Caution	2015-05-25 10:22:43	173
Caution	2015-08-02 15:47:10	221

When an Outage is current.

A dimension is used to **search** for data rows from a fact table and with the view System_Current_Unique presents in a dropdown box search criteria.

System_dkey	System
19	Caution

It is easy inside a BI tool to make a dimension for recent values and join it to the fact through the durable key column System_dkey in the BI tool's data model. A BI tool shows the most recent values of System e.g. Caution and a business users select a value in a dropdown box search criteria, and a BI tool does a join to a fact table durable key to fetch all data rows that are using the select system even though the name of the system has been changed over time:

```
SELECT d.System, f.OutageDate, f.ProductionKWh
FROM   fact.Outage f
       INNER JOIN dim.System_Current_Unique d ON d.System_dkey = f.System_dkey
WHERE d.System = 'Caution'
```

System	OutageDate	ProductionKWh
Caution	2015-05-25 10:22:43	173
Caution	2015-08-02 15:47:10	221

Original view with dimension key

```
CREATE VIEW dim.System_Original AS
SELECT System_dkey, System
FROM   dim.System
WHERE System_pkey = 0
```

Used in join with fact table to fetch and see original value for a fact row:

```
SELECT d.System, f.OutageDate, f.ProductionKWh
FROM   fact.Outage f
       INNER JOIN dim.System_Original d ON d.System_dkey = f.System_dkey
```

System	OutageDate	ProductionKWh
Warning	2015-05-25 10:22:43	173
Warning	2015-08-02 15:47:10	221

When an Outage was original.

Historical function with a datetime parameter with dimension key

```
CREATE FUNCTION dim.System_Historical(@ReportTime datetime)
RETURNS TABLE AS RETURN (
SELECT System_dkey, System
FROM   dim.System
WHERE @ReportTime >= ValidFrom AND @ReportTime < ValidTo)
```

Used in join with fact table to fetch and see historical value for a fact row at a reporting datetimestamp, e.g. 2015-06-01 10:24:

```
SELECT d.System, f.OutageDate, f.ProductionKWh
FROM   fact.Outage f
       INNER JOIN dim.System_Historical('2015-06-01 10:24') d
         ON d.System_dkey = f.System_dkey
```

System	OutageDate	ProductionKWh
Notification	2015-05-25 10:22:43	173
Notification	2015-08-02 15:47:10	221

When an Outage is point-in-time.

BI tools like Excel, Power BI, Tableau, QlikView, Qlik Sense, Alteryx or Targit does not yet offer a business user to type-in a reporting datetimestamp for a dimension before showing the values of the dimension, but a report has normally some criteria boxes and one of them could be an input datetimestamp for a historical dimension, so a report shows the values from a specific point-in-time, maybe to reproduce exactly the same report as it was made before at a given reporting datetimestamp.

The above views is in a database schema dim, and name of a view has a suffix as: _Current, _Current_Unique, _Historical, _Original or _Registered.

To separate tables and views in different database schema, I like to use database schema dim and fact for tables, and database schema dims and facts for views. Business users have granted read access permission to the views, not the tables.

An alternative for the above views is to place them in four database schemas: dimcurrent, dimhistorical, dimoriginal and dimregistered and reuse name of view System in all schemas plus a System_Unique in dimcurrent schema.
Writing a sql statement Inner Join part will use IntelliSense e.g. dimcurrent. and get a list of all dimensions that presents current values in join with a fact table.

6.3.4. Conclusion for a type 7 dimension

Dimension type 7 provides up to five values for registered, current (in a non-unique list), current (in a unique list), original and historical and it is very flexible for a fact data row to request for and to fetch a dimension value.

Of course a lot of fact data rows have only one value in a dimension, like a fact for sale to a new customer that is at the same time entered into the dimension. When the customer don't change any data for a long time, the dimension value is registered = current = original = historical.

The customer purchases many times and have many rows in Sales fact, but one day the customer moves from his home country to three different countries over time and still continue to purchase. This means that the original value becomes the home country, historical values are the second and third country, and current value is the fourth and recent country, and the registered value country is depending of the date of sale.

For statistics of the sales we can have summaries for customers home countries even though they have moved to other countries because we simple use the original value of the dimension. We can have summaries for countries where the customer did their purchase from by using the registered value.

In case we like to know the sales from customers who have once lived in a specific country, we first lookup that country in a dimension customer current unique view and let column _dkey join to the sales fact table column _dkey.

The marketing people like to know the current address, city, phone number and email of all the customers for a marketing planning to contact them with new exciting offers in a marketing campaign.

Type 7 gives you all the options and the five helper sql views is handy in a BI tool data model and for ad hoc queries in SQL.

The right information, at the right time, in the right place, in the right way, in the right format, to the right person.

In a data access BI tool it is easy to make dimensions based on the helper views and join the dimensions to the fact through the key or dkey columns in the BI tool's data model.

Type 7 is better than type 2 because the current unique view upon a type 7 dimension will show a **unique list** of distinct current values which is very handy for a dropdown box search criteria for a report and for a dashboard in a BI tool.

6.3.5. Extra columns for a type 7 dimension table and a fact table

A dimension table in the physical design will have **extra key columns**:

- **_key** used for registered value of a dimension.
- **_dkey** used for current, original and historical value of a dimension.
  (stands for a durable key as an integer representation of a business key and comes from a key mapping table)
- **_pkey** used for previous key that is a reference to the _key column.
- **_bkey** used for business key from a source system.
- **_skey** used for surrogate key from a source system.
- **_sbkey** used for surrogate business key which is a unique code.

A fact table in the physical design has for each dimension two columns:

- **_key** used for registered value and current value of a dimension.
- **_dkey** used for current, original and historical value of a dimension.

A type 7 fact table contains dual foreign keys for a given dimension:

- A dimension key _key to join on a dimension Registered view to fetch the registered values for a fact row, and to join on a dimension Current view to fetch the current values for a fact row.

- A durable key _dkey to join on a dimension Current unique view to fetch the current values for a fact row, which is good for dropdown box search criteria for a report and for a dashboard in a BI tool because of the unique list.

Datetimestamp in dimension
A dimension table will keep track of datetimestamp of inserting and updating a row with two extra audit columns:

- **InsertTime** database server system time when the row was inserted.
- **UpdateTime** database server system time when the row was updated.

I don't like an audit column to be Null or N/A, therefore UpdateTime gets the same datetimestamp as InsertTime, so both columns will have the same value until data in the row will be either changed or deleted in a source system. The two audit columns can be assigned either by triggers for insert and update at the table or by a ETL process. I prefer a ETL process to handle all audit columns.

InsertTime is a recording time of when data was recorded in a data warehouse.
UpdateTime is a changing time of when a value was changed in a data warehouse.
OrderDate, BirthDate or TransactionTime, etc. from a source system at the time a transaction was happening, occurred, registered or took place, a happened time of when an event happened.

Execution stamp in dimension
A dimension table will keep track of job, package or data flow execution log identification stamp of inserting and updating a row with two audit columns:

- **InsertExecutionLogId** execution identification that has inserted the row.
- **UpdateExecutionLogId** execution identification that has updated the row.

Execution log identification or Process Id is useful for an audit trail to do a log of number of rows of a data batch that has been inserted or updated into a table in the data warehouse.

A dimension table will keep track of a **current value** in a source system with a metadata column which is a boolean (bit) with two values True (1) and False (0):

IsCurrent in dimension to mark current dimension data values
IsCurrent = 1 when the row represents the current value in a source system.
IsCurrent = 0 when the row represents a historical value from a source system.

Column IsCurrent can also be called CurrentRowIndicator, ActiveFlag, RecentFlag, ActiveRecord or ActiveRow, etc.

When IsCurrent = 1 the timeline medata columns ValidFrom and ValidTo represents the time period or time span where a row value is current in a source system.

A dimension table will keep track of an **expired or deleted value** in a source system with a metadata column which is a boolean (bit) with two values True (1) and False (0):

IsDeleted in dimension to handle deletion of data in a source system
IsDeleted = 1 when the row represents the deleted value from a source system.
IsDeleted = 0 when the row represents an existing value in a source system.

Column IsDeleted can also be called DeleteFlag, DeletedAtSource, ExpiredFlag or InactiveFlag, when data is deleted in a system and therefore are gone forever.

When IsDeleted = 1 the timeline medata columns ValidFrom and ValidTo represents the time period or time span where a row value does not exists in a source system because of a hard delete or a soft delete in a source system and told to data warehouse.

IsCurrent = 1 and IsDeleted = 1 in dimension to handle deletion of data
A row in a dimension can be both IsCurrent = True and IsDeleted = True, because the row is current for the old fact rows and the row is flagged deleted so a future new fact row can not refer to the row because the business key of the row is not available in a source system, which a ETL process will handle. Therefore a fact table data is consistent (not inconsistent) and will never be missing a dimension value in a dimension table.

RowChangeReason in dimension
A dimension table can keep track of the reason why the row was changed in a column called RowChangeReason so it is easy for doing search and analysis. The column could have value »New« when the row represents a new customer, and later when customer move to another country the value could change to »Moved country« and later it would be »Changed credit status« and so on.

Datetimestamp in fact
A fact table will keep track of datetimestamp of inserting a row with two extra audit columns:

- **InsertTime** database server system time when the row was registered.
- **UpdateTime** database server system time when the row was updated.

When there is no transaction date for fact data, the InsertTime will be used to date range lookup a unique dimension table row. (Other terms can be Receipt time, Registered time, Recording time or Load time/Load date but not Happening time, Appearance time, Business time, Transaction time, Entry time, Positing time or Posting time (in danish Bogføringstid) from the real world).

A fact table will keep track of datetimestamp of updating a row with extra metadata columns:

- **IsCurrent** if fact row will change then keep track of current row.
- **IsDeleted** if fact row can be deleted in a source system.
- **BeginAt** if fact row is treated as SCD type 2, Slowly Changing Facts.
- **EndAt** if fact row is treated as SCD type 2, Slowly Changing Facts.

BeginAt date and EndAt date (or datetimestamp) timeline metadata columns for each fact row is to track the versions of a fact and allowing point-in-time analysis.

Slowly Changing Facts SCF or Frequently Changing Facts FCF but hopefully not Rapidly Changing Facts RCF because that could give a lot of facts rows over time with BeginAt and EndAt timelines.

When transactions are changed retroactively, you should update corresponding fact rows. If you need to track the history of fact rows it becomes Slowly Changing Facts, read more in section 6.4.

Execution stamp in fact
A fact table will keep track of job, package or data flow execution log identification stamp of inserting and updating a row with two audit columns:

- **InsertExecutionLogId** execution identification that has inserted the row.
- **UpdateExecutionLogId** execution identification that has updated the row.

Execution log identification or Process Id is useful for an audit trail to do a log of number of rows of a data batch that has been inserted or updated into a table in the data warehouse.

6.3.6. Extended example of a type 7 dimension table and a fact table

Let's have an example where a source system has changed its value three times from »Warning« to »Notification« and to »Caution« and in the ending the source system will delete the value so it will not exists anymore, but the dimension will keep it forever. Each of the following steps will show how the dimension table looks like after each process has been done.

1. First time a value arrives to the data warehouse which happen at Maj 1, 2015, 5 pm or 17:00, a new row is added to the dimension table to contain the new value in column System together with key, audit, and metadata columns.

_key	_dkey	_pkey	_bkey	System	ValidFrom	ValidTo	IsCurrent	IsDeleted	InsertTime	UpdateTime
19	19	0	76CB	Warning	1900-01-01 00:00:00	9999-12-31 23:59:59	1	0	2015-05-01 17:00:00	2015-05-01 17:00:00

2. Value changed at 2015-05-08 therefore the first row is not current and a new row is added.

_key	_dkey	_pkey	_bkey	System	ValidFrom	ValidTo	IsCurrent	IsDeleted	InsertTime	UpdateTime
19	19	0	76CB	Warning	1900-01-01 00:00:00	2015-05-08 14:12:21	0	0	2015-05-01 17:00:00	2015-05-08 17:01:09
35	19	19	76CB	Notification	2015-05-08 14:12:21	9999-12-31 23:59:59	1	0	2015-05-08 17:01:10	2015-05-08 17:01:10

3. Value changed at 2015-07-31 therefore the second row is not current and a new row is added.

_key	_dkey	_pkey	_bkey	System	ValidFrom	ValidTo	IsCurrent	IsDeleted	InsertTime	UpdateTime
19	19	0	76CB	Warning	1900-01-01 00:00:00	2015-05-08 14:12:21	0	0	2015-05-01 17:00:00	2015-05-08 17:01:09
35	19	19	76CB	Notification	2015-05-08 14:12:21	2015-07-31 08:45:48	0	0	2015-05-08 17:01:10	2015-07-31 14:03:22
87	19	35	76CB	Caution	2015-07-31 08:45:48	9999-12-31 23:59:59	1	0	2015-07-31 14:03:23	2015-07-31 14:03:21

4. Value deleted at 2015-08-06 where the current row will be copied to a new row and will be marked deleted to handle deletion of data in a source system.

_key	_dkey	_pkey	_bkey	System	ValidFrom	ValidTo	IsCurrent	IsDeleted	InsertTime	UpdateTime
19	19	0	76CB	Warning	1900-01-01 00:00:00	2015-05-08 14:12:21	0	0	2015-05-01 17:00:00	2015-05-08 17:01:09
35	19	19	76CB	Notification	2015-05-08 14:12:21	2015-07-31 08:45:48	0	0	2015-05-08 17:01:10	2015-07-31 14:03:22
87	19	35	76CB	Caution	2015-07-31 08:45:48	2015-08-06 10:10:05	0	0	2015-07-31 14:03:23	2015-08-06 12:00:06
99	19	87	76CB	Caution	2015-08-06 10:10:05	9999-12-31 23:59:59	1	1	2015-08-06 12:00:08	2015-08-06 12:00:05

With this approach there will be inserted an additional row for a deleted (or closed item) from a source system, and a dimension will tell in the last row with marked IsDeleted = 1, that the business key has been deleted in source system with a period from 2015-08-06 10:10:05 to forever. IsDeleted = 1 represents a deleted row with a business key value that is no longer available in a source system in a period of ValidFrom/ValidTo.

Previous row with IsCurrent = 0 represents an old historical version of a dimension value and ValidFrom/ValidTo tells us the period when the old value was current or active back in time. Older fact row can be referring to older dimension key values.

No fact row with a dimension key in column _key will never refer to a mark deleted row in a dimension, therefore no fact row can use a dimension key value in column _key with IsDeleted = 1.

Therefore a fact table data is consistent (not inconsistent) and will never be missing a dimension value in a dimension table. In case a deleted business key value will be reopen (reappear or reborn) with the same business key valu in a source system, the current dimension row will be updated with IsCurrent = 0 and a ValidTo datetime, and a new row is inserted and becomes current, and we keep IsDeleted = 1 to show the value was deleted in a previous period.

When source data with an OutageDate arrives to the data warehouse it will find the historical system name at the time when an outage was happening with a criteria that will not include the deleted dimension value because fact data with a _key column can't refer to a deleted value:

```
IsDeleted = 0 AND System_bkey = SystemId AND
OutageDate >= ValidFrom AND OutageDate < ValidTo
```

Example of two source data rows.

OutageDate	SystemId	ProductionKWh
2015-05-25 10:22:43	76CB	173
2015-08-02 15:47:10	76CB	221

They become two fact data rows where System_key is a reference to the System dimension value at the registered fact time from column OutageDate and System_dkey is a reference to the entity with business key 76CB to provide current, original or historical value.

OutageDate	System_key	System_dkey	ProductionKWh
2015-05-25 10:22:43	35	19	173
2015-08-02 15:47:10	87	19	221

The current view for the dimension to show all current values from the dimension will search after criteria IsCurrent = 1, so the view gives the current values every time I access the view, and the fact column _dkey can join to the view for showing the current values of the dimension, therefore dimension values with IsDeleted = 1 will be included in the view because there are old fact data back in time with a _dkey which refers to deleted dimension data in the source system.

Helper views
It is hard to remember the criteria for _key, _dkey, ValidFrom and ValidTo therefore I will make sql helper views to do a join between dimension and fact more easy for business users and analysis tools.

View provide registered dimension values and will be joined to fact data through _key

```
CREATE VIEW dim.System_Registered AS
SELECT System_key, System
FROM   dim.System
WHERE IsDeleted = 0
```

System_key	System
19	Warning
35	Notification
87	Caution

Please notice, that the dimension rows with marked IsDeleted = 1, because the business key has been deleted in source system, is not included in the view because no fact data refer to them.

View provide current dimension values and will be joined to fact data through _key

```
CREATE VIEW dim.System_Current AS
SELECT d.System_key, c.System
FROM   dim.System d
        INNER JOIN  -- self join for a type 7 dimension to fetch current values.
          (SELECT System_dkey, System  -- the most recent System.
           FROM dim.System
           WHERE IsCurrent = 1
          ) c ON c.System_dkey = d.System_dkey
WHERE d.IsDeleted = 0
```

System_key	System
19	Caution
35	Caution
87	Caution

Please notice, that the dimension rows with marked IsDeleted = 1, because the business key has been deleted in source system, is not included in the view because no fact data refer to them.

View provide unique current dim. values and will be joined to fact data through _dkey

```
CREATE VIEW dim.System_Current_Unique AS
SELECT System_dkey, System
FROM   dim.System
WHERE IsCurrent = 1
```

System_dkey	System
19	Caution

View provide original dimension values and will be joined to fact data through _dkey

```
CREATE VIEW dim.System_Original AS
SELECT System_dkey, System
```

```
FROM   dim.System
WHERE System_pkey = 0
```

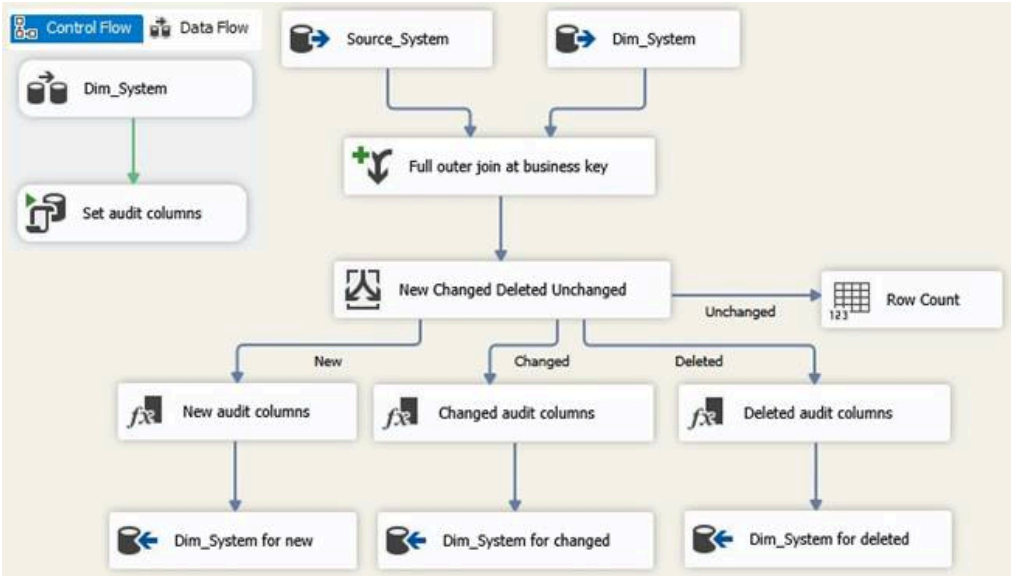
System_dkey	System
19	Warning

View provide historical dimension values and will be joined to fact data through _dkey

```
CREATE FUNCTION dim.System_Historical(@ReportTime datetime)
RETURNS TABLE AS RETURN (
SELECT System_dkey, System
FROM   dim.System
WHERE IsDeleted = 0 AND @ReportTime >= ValidFrom AND @ReportTime < ValidTo)
```

When dimension data has IsDeleted = 1 in a new dimension row, the value of _key is never used among fact data _key column, therefore the registered view and the historical view does not need to include deleted dimension data. Anyway the same dimension data exists already in the previous row in the dimension table with a ValidFrom/ValidTo period for the time when the data was current and active in a source system.

An implementation in SSIS SQL Server Integration Services of type 7 dimension



Read how to implement the type 7 dimension ETL process in SSIS and how to load fact data with a transaction date and in a steaming ETL way with incremental load or delta load

Example of a stored procedure to query to fetch data from a type 7 star schema

```
CREATE PROCEDURE dbo.Banktransaction
 -- Default value indicates a parameter must not be null in the call.
 @FromDate       date = '2014-01-01',
 @ToDate         date = '2014-12-31',
 @CustomerNumber int, -- a business key for a customer, where null means all customers.
 @AccountTypeCodeList nvarchar(MAX) = N'STANDARD,PENSION,HOUSE,CAR,DEPOSIT'
AS                                  -- a surrogate business key for types of accounts.
BEGIN
 SET NOCOUNT ON
 DECLARE @Customer_dkey int -- durable key for the business key.
 SELECT @Customer_dkey = Customer_dkey
 FROM DMA.dim.Customer_Current_Unique -- from database DMA and schema dim.
 WHERE CustomerNumber = @CustomerNumber

 SELECT
   CustomerName = dc.CustomerName,
   BankDate     = fb.TransactionDate,
   Account      = fb.AccountNumber,
   Product      = dp.ProductName
 FROM DMA.fact.Banktransaction fb  -- from database DMA and schema fact.
     INNER JOIN DMA.dim.Customer_Current dc ON dc.Customer_key = fb.Customer_key
     INNER JOIN DMA.dim.Product_Registered dp ON dp.Product_key = fb.Product_key
 WHERE fb.TransactionDate BETWEEN @FromDate AND @ToDate AND
  ( (@CustomerNumber IS NULL) OR (fb.Customer_dkey = @Customer_dkey) ) AND
    (fb.AccountType_dkey IN (SELECT AccountType_dkey FROM DMA.dim.AccountType_Current_Unique
         WHERE AccountTypeCode IN (SELECT value FROM string_split(@AccountTypeCodeList, ','))))
 ORDER BY 1,2,3
 OPTION(RECOMPILE) --build query execution plan based on parameter value from the call (runtime).
END
```

Dimensions of Kimball type 7 is shown in INNER JOIN and WHERE parts:

INNER JOIN is using a Current view to fetch Kimball type 1 column for SELECT to show the most recent, current value, above the name of the customer.

INNER JOIN is using a Registered view to fetch Kimball type 2 column for SELECT to show the registered value at the time the bank transaction happened, above the name of the product, because these names happens to change over time.

WHERE is using durable key _dkey column to find the current AccountType e.g. STANDARD even though the term was NORMAL in the first half of year 2014.
We can use both views: dim.AccountType_Current_Unique or dim.AccountType_Current.

A dimension table can have a metadata column like IsInferred as a boolean (bit) with two values True (1) and False (0):

IsInferred in dimension to handle early arriving fact
IsInferred = 1 when the row represents a business key value from a source system with no descriptive data values that become an unknown in dimension.
IsInferred = 0 when the row represents a known value in a source system.

An example with artifact values or inferred members
Let's have an example where a source system has business key with null and unknown values that does not exists in the dimension table. It is handle as inferred members -1 missing and new inferred members with column IsInferred value 1.

System_key	System_dkey	System_pkey	System_bkey	System	ValidFrom	ValidTo	IsCurrent	IsDeleted	IsInferred
-1	-1	0	0	Missing	1900-01-01 00:00:00	9999-12-31 23:59:59	1	0	1
19	19	0	76CB	Warning	1900-01-01 00:00:00	2015-05-08 14:12:21	0	0	0
35	19	19	76CB	Notification	2015-05-08 14:12:21	2015-07-31 08:45:48	0	0	0
87	19	35	76CB	Caution	2015-07-31 08:45:48	9999-12-31 23:59:59	1	0	0
88	88	0	88FF	Sign	1900-01-01 00:00:00	9999-12-31 23:59:59	1	0	0

Example of table SourceProduction with business key called SystemId.

OutageDate	SystemId	ProductionKWh
2015-05-25 10:22:43	76CB	173
2015-08-02 15:47:10	76CB	221
2015-08-12 22:21:42	88FF	100
2015-10-02 00:00:00	90AA	200
2015-10-12 00:00:00	90AA	300
2015-10-15 00:00:00	91KL	400
2015-02-07 00:00:00	NULL	500

All data rows in table SourceProduction will be loaded to table FactSystem where the business key SystemId will be transformed to the dimension key System_key in table DimSystem. There is three values in SystemId (90AA, 91KL, NULL) in four rows that does not exists in DimSystem, therefore they are inferred members and is infer to DimSystem where NULL will be transformed to System_key -1 for »Missing«, and 90AA and 91KL will be inserted into new data rows in DimSystem as »Unknown« with new numbers in column System_key with IsInferred value 1 and they will be used in loading data rows into FactSystem table.

After running a ETL process the table DimSystem will have new inferred rows.

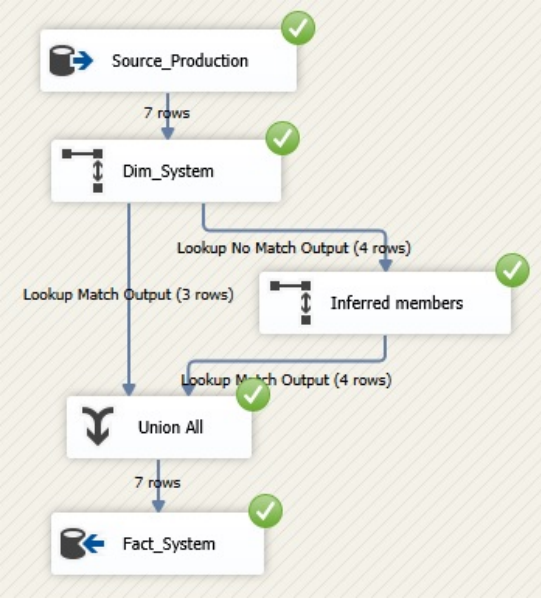
System_key	System_dkey	System_pkey	System_bkey	System	ValidFrom	ValidTo	IsCurrent	IsDeleted	IsInferred
-1	-1	0	0	Missing	1900-01-01 00:00:00	9999-12-31 23:59:59	1	0	1
19	19	0	76CB	Warning	1900-01-01 00:00:00	2015-05-08 14:12:21	0	0	0
35	19	19	76CB	Notification	2015-05-08 14:12:21	2015-07-31 08:45:48	0	0	0
87	19	35	76CB	Caution	2015-07-31 08:45:48	9999-12-31 23:59:59	1	0	0
88	88	0	88FF	Sign	1900-01-01 00:00:00	9999-12-31 23:59:59	1	0	0
89	89	0	90AA	Unknown 90AA	1900-01-01 00:00:00	9999-12-31 23:59:59	1	0	1
90	90	0	91KL	Unknown 91KL	1900-01-01 00:00:00	9999-12-31 23:59:59	1	0	1

The table FactSystem is loaded with all the data rows from SourceProduction and is also using the new numbers in column System_key from the table DimSystem.

OutageDate	System_key	System_dkey	ProductionKWh
2015-05-25 10:22:43	35	19	173
2015-08-02 15:47:10	87	19	221
2015-08-12 22:21:42	88	88	100

2015-10-02 00:00:00	89	89	200
2015-10-12 00:00:00	89	89	300
2015-10-15 00:00:00	90	90	400
2015-02-07 00:00:00	-1	-1	500

An implementation in SSIS SQL Server Integration Services of Inferred members



Read how to implement Inferred members in a SSIS Lookup

6.4. Slowly Changing Facts

Versioning fact rows, can a fact table be versioned?
Most of the time a fact table row will never be updated but when it is happening I will keep history of a measure value in a fact table row by inserting a new row so the fact table changes are treated as SCD type 2. The fact data is provided with a business key (composed of several columns) from a source system, so the data warehouse knows when the data has been changed. There will be a »between-filter« in the reporting tool on the fact table.

6.4.1. Example of capture a relationship in the Timespan factless fact

An example is an operational system that contains current relationship of which buildings house which departments are located. Over time a department will relocate to another building.
When the two dimensions is type 1 we use the dimension key _key.
When the two dimensions is type 2 we use the durable key _dkey.

When we capture a relationship in the fact, it will include history of previous locations with a time period or time span of BeginAt date and EndAt date, where BeginAt date is one day after the previous EndAt date. With no overlap there can never be any confusion of the span of time.
Every time a department changes status it become a new row in a fact table with a status.

Building_key	Department_key	Status_Key	BeginAt	EndAt	IsCurrent
1	1	Relocated	2010-01-01	2011-03-31	0
1	2	Relocated	2010-01-01	2012-02-09	0
1	3	Located	2010-06-15	9999-12-31	1
2	1	Relocated	2011-04-01	2012-12-31	0
2	4	Relocated	2011-05-01	2012-12-31	0
3	2	Located	2012-02-10	9999-12-31	1
3	1	Located	2013-01-01	9999-12-31	1
3	4	Closed	2013-01-01	2014-12-31	1
4	1	Located	2013-01-01	9999-12-31	1

The factless fact is called a timespan fact or history fact or historical fact or Status fact.

For better reading I have for dimension key Status used text instead of key number value.
Building key 2 has no more departments because the building burned down on New Year's Eve to the year 2013. Department key 1 was from year 2013 located in two buildings (key 3 and 4). Department key 4 was closed at the end of 2014.

In a data vault model we will have two tables:
LinkLocation (BuildingHashKey, DepartmentHashKey, StatusHashKey)
SatLocation (LocationHashKey, BeginAt, EndAt, IsCurrent) as an Effectivity satellite.

6.4.2. Example of Timespan fact or history fact or historical fact or Status fact

A marketing system contains the expected sale for each sales person and each year, example for year 2016 Mr. Sellers expects to have a sales of 10 quantity to a budget amount of 100. But as the year goes by the expectation will be changed to adapted the marked and the figures will be overwritten in the system

together with a status that's starts at open for changes and later closed for changes, or canceled if the sales person failed in his duty. By overwritten it means it is a task for the data warehouse fact table to remember all figures and statuses like it was a type 2 dimension that is using row versioning (compliance-enabled), that means the fact table has BeginAt and EndAt timestamps, where BeginAt date is one day after the previous EndAt date.

Every time a person changes its status and measure data it become a new row in a fact table.

Year	Person	Quantity	Budget	Status	BeginAt	EndAt	IsCurrent
2016	005 Sellers	10	100	Open	2015-12-12	2016-03-26	0
2016	005 Sellers	11	125	Open	2016-03-27	2016-04-12	0
2016	005 Sellers	9	117	Open	2016-04-13	2016-06-20	0
2016	005 Sellers	14	130	Open	2016-06-21	2016-07-31	0
2016	005 Sellers	15	130	Closed	2016-08-01	9999-12-31	1
2016	007 Bond	10	100	Open	2015-12-16	2016-03-31	0
2016	007 Bond	20	180	Open	2016-04-01	2016-05-20	0
2016	007 Bond	0	0	Canceled	2016-05-21	9999-12-31	1

Measure Quantity and Budget is Semi-Additive.

Dimensions will be for Year, Person, Status.
A requested date like 2016-04-15 in SQL: Where '2016-04-15' Between BeginAt And EndAt.
A requested year like 2016 in SQL: Where BeginAt <= '2016-12-31' And EndAt >= '2016-01-01'.
A requested count of number of the days each person has status open in 2016 in SQL:
Select Person, Days = Sum(Least('2016-12-31', EndAt) - Greatest('2016-01-01', BeginAt))
From fact
Where Status = 'Open' And BeginAt <= '2016-12-31' And EndAt >= '2016-01-01'
Group By Person.

6.4.3. Example of Counterpart fact

Financial accounting counterpart, where IsCp = is counterpart = 1 with an entry date into a fact table. A counterpart has its own row before the changed data in a new row.

Year	Person	Quantity	Budget	Status	EntryDate	IsCp	IsCurrent	FactDataId
2016	005 Sellers	10	100	Open	2015-12-12	0	0	123
2016	005 Sellers	-10	-100	Open	2016-03-27	1	0	123
2016	005 Sellers	11	125	Open	2016-03-27	0	0	123
2016	005 Sellers	-11	-125	Open	2016-04-13	1	0	123
2016	005 Sellers	9	117	Open	2016-04-13	0	0	123
2016	005 Sellers	-9	-117	Open	2016-06-21	1	0	123
2016	005 Sellers	14	130	Open	2016-06-21	0	0	123
2016	005 Sellers	-14	-130	Open	2016-08-01	1	0	123
2016	005 Sellers	15	130	Closed	2016-08-01	0	1	123
2016	007 Bond	10	100	Open	2015-12-16	0	0	124
2016	007 Bond	-10	-100	Open	2016-04-01	1	0	124
2016	007 Bond	20	180	Open	2016-04-01	0	0	124
2016	007 Bond	-20	-180	Open	2016-05-21	1	0	124
2016	007 Bond	0	0	Canceled	2016-05-21	0	1	124

Measure Quantity and Budget is Fully-Additive.

I will use EntryDate to find a month calculating summary, e.g. April 2016, and it does not care of the expectations that has been changed after April because EntryDate is a time machine dimension used to go back in time and make a roll up aggregation:

```
SELECT Person, Quantity = SUM(Quantity), Budget = SUM(Budget)
FROM    Fact
WHERE  Year = 2016 AND EntryDate <= '2016-04-30'
GROUP BY Person
```

With data for some years and loaded into a OLAP cube with a analysis tool I select a given date in Entry date dimension, e.g. 2016-04-30, and I use the Year dimension as column in a matrix to compare the month of April between the years.

Year	2016		2015		2014		2013	
Person	Budget	Index	Budget	Index	Budget	Index	Budget	Index
005 Sellers	117	104	112	113	99	132	75	112
007 Bond	180	200	90	290	31	148	21	97

[An extended technique when there can be multiple changes over time, is to extract and negate all versions of the fact rows into a holding table (old positives become negative and old negatives become positive), with the addition of a positive version of the new row. Then summarize the holding table across all dimensions (except activity/posting date, use max() if it is in the holding table), and filter out zero rows, the result is the total net change resulting in one or two rows per fact. You would wind up with two rows if there is a change in one or more dimensions, one row if the dimensions did not change and no rows if there was no change (all measures are zero).]

6.4.4. Example of Transactional fact

Financial transactions net change, where IsOriginal = 1 contain the original values and
IsOriginal = 0 contain the net change or the delta values.

Year	Person	Quantity	Budget	Status	EntryDate	IsOriginal	FactDataId
2016	005 Sellers	10	100	Open	2015-12-12	1	123
2016	005 Sellers	1	25	Open	2016-03-27	0	123
2016	005 Sellers	-2	-8	Open	2016-04-13	0	123
2016	005 Sellers	5	13	Open	2016-06-21	0	123
2016	005 Sellers	1	0	Closed	2016-08-01	0	123
2016	007 Bond	10	100	Open	2015-12-16	1	124
2016	007 Bond	10	80	Open	2016-04-01	0	124
2016	007 Bond	-20	-180	Canceled	2016-05-21	0	124

Measure Quantity and Budget is Fully-Additive.

A transactional fact table is insert only and stores changes as the difference or the deviation between the current state and the new state.

We can also keep the original columns which show current values at EntryDate, and then we add two extra columns with net change in QuantityDelta and BudgetDelta.

I can use the same sql statement as above and this fact table approach keep the same number of rows as the original fact table because there is no counterpart rows, but the loading needs to do some calculations, there is always pros and cons with a technique.

6.4.5. Example of Timespan accumulating snapshot fact

Accumulating snapshot fact is extended with SCD type 2 as a periodic snapshot fact to handle a business process with both stages and states or statuses to fulfills Inmon's data warehouse definition for non-volatile. I wrote about in it section 3.3 types of fact tables and it is based on Kimball Design Tip #145.

6.5. Kimball type 2 dimension, ValidFrom date, implement by Merge

Kimball type 2 dimension how to determine a date for timeline metadata columns ValidFrom and ValidTo (Effective date and Expiration date), and how to implement a SQL Merge statement for a type 2 dimension with artifact values or inferred members and correction of data.

6.5.1. SCD Type 1 versus SCD Type 2

When we choose a dimension to be type 1, it is because we don't want to keep the old values. We want only current values for the lifetime of a data warehouse. For example »types of employee skills« because old names of the types are not wanted in a report or a dashboard, or a car factory specification will never change except for a correction of data values. A type 1 dimension wants to present values at any time as current data and »one truth«, therefore old values are not important.

When we choose a dimension to be type 2, it is because we do want to keep the old values. We want registered values for the lifetime of a data warehouse, because we want to present data values as they were when fact data was happening, occurred, registered or took place in the operational systems and it is often with a transaction date or an event date else when data was entered into the fact. For example »addresses of customers' residences« because old addresses and cities are wanted in a report or a dashboard back in time for statistic of where the customers lived at date of purchase. A type 2 dimension wants to present values at any time as current data back then and »current truth«, therefore old values are important.

Type 7 does same as Type 2, but the list of current values are unique with Type 7 because of the dimension durable key in fact together with dimension key.

6.5.2. Type 2 ValidFrom determination

A type 2 dimension has timeline metadata columns ValidFrom and ValidTo which represents the span of time when a row in a dimension was the »current truth« or when the data values was/is valid in an active period. It is the same for a type 7 dimension.

Kimball calls it »precise time stamping of a type 2 slowly changing dimension« because to determine a new value in a ValidFrom column in a type 2 dimension must be an »exact date/time of change«.
I call it Kimball type 2 ValidFrom determination.

Lets say we have a type 2 Customer dimension with Marie Beaulieur as her maiden name was valid from she was born in Lyon or from she becomes a customer at 2005-12-01 and that date is known in a source system. The first entering will use a beginning of time 1900-01-01 to determine a default value in a ValidFrom column, and use a end of time 9999-12-31 as default value in a ValidTo column.

Customer_key	Customer-Id_bkey	CustomerName	City	ValidFrom	ValidTo
3	421	Marie Beaulieur	Lyon	1900-01-01	9999-12-31

And a Sales fact with dimension key columns to dimensions of type 2.

PurchaseDate_key	Customer_key	Product_key	Quantity	Amount

2005-12-01	3	32	2	200
2006-12-05	3	17	1	65
2007-09-12	3	32	3	300

PurchaseDate is the transaction date, when a purchase was happening, occurred, registered or took place. PurchaseDate_key is a role-playing date dimension with the role »purchase«.

At date 2007-11-16 Marie Beaulieur got married, took her husband's surname to Marie Lesauvage and moved to Paris.

We don't know anything about it before she does a new purchase at 2007-12-21 as a business date to determine a new value in a ValidFrom column.

Customer_key	Customer-Id_bkey	CustomerName	City	ValidFrom	ValidTo
3	421	Marie Beaulieur	Lyon	1900-01-01	2007-12-21
87	421	Marie Lesauvage	Paris	2007-12-21	9999-12-31

When an operational system requests a user for a date to record or register when the data values have been changed, e.g. Marie's wedding date at 2007-11-16 as a business date to determine a new value in a ValidFrom column.

Customer_key	Customer-Id_bkey	CustomerName	City	ValidFrom	ValidTo
3	421	Marie Beaulieur	Lyon	1900-01-01	2007-11-16
87	421	Marie Lesauvage	Paris	2007-11-16	9999-12-31

When we want to store a business date, we can keep it in one column or we can extend Customer dimension with two extra timeline columns BusinessFrom and BusinessTo:

Customer_key	Customer-Id_bkey	Customer Name	City	Business From	Business To	ValidFrom	ValidTo
3	421	Marie Beaulieur	Lyon	2005-12-01	2007-11-15	1900-01-01	2007-11-16
87	421	Marie Lesauvage	Paris	2007-11-16	<null>	2007-11-16	9999-12-31

BusinessFrom and BusinessTo represents the span of time when a row in a dimension was the »current truth« with criteria Between BusinessFrom And BusinessTo.

ValidFrom and ValidTo represents the span of time when a row in a dimension was the »current truth« with criteria >= ValidFrom And < ValidTo.

First time a business key value enters a type 2 dimension, a ETL process will insert a new row with ValidFrom 1900-01-01, therefore good idea to have a BusinessDate column with the original date from a source system, e.g. 2005-12-01.

When we transfer a purchase into a fact table to fetch a dimension key value, it will do a date range lookup with this criteria:

```
CustomerId_bkey = CustomerId AND
PurchaseDate >= ValidFrom AND PurchaseDate < ValidTo
```

Sales fact with dimension key columns to dimensions of type 2 where the new purchase at 2007-12-21 refers to Customer_key value 87 for Marie Lesauvage living in Paris, and the old purchase rows are still referring to Customer_key value 3 for Marie Beaulieur lived in Lyon.

PurchaseDate_key	Customer_key	Product_key	Quantity	Amount
2005-12-01	3	32	2	200
2006-12-05	3	17	1	65
2007-09-12	3	32	3	300
2007-12-21	87	17	2	130

A »monthly sales in cities« report will use the city of the customers at the time a purchase was happening, occurred, registered or took place.

Often an operational system has no business date, only a transaction date or an event date, e.g. date of purchase. Therefore it is very important for a type 2 dimension to determine a new value in a ValidFrom column to be set with the right date when a source system data value has been changed.

When we receive a data set including data about customers and their purchases at a date of purchase, we can assume together with the operational system and the business people, that, if a customer has different data values compared to a type 2 dimension current row with ValidTo 9999-12-31 or IsCurrent = 1, then a new row will be inserted into a Customer dimension with ValidFrom set to date of purchase.

Sometimes an operational system has no business date and no transaction date and no event date. When a ETL process is running **after midnight** it could determine a new value in a ValidFrom column as current date value, e.g. 2007-12-22.

Customer_key	Customer-Id_bkey	CustomerName	City	ValidFrom	ValidTo
3	421	Marie Beaulieur	Lyon	1900-01-01	2007-12-22
87	421	Marie Lesauvage	Paris	2007-12-22	9999-12-31

The date range lookup for the new purchase at 2007-12-21 will refer to Customer_key value 3 for Marie Beaulieur living in Lyon, which is very wrong, because it was at 2007-11-16 Marie got married, took her husband's surname Lesauvage and moved to Paris.

When we receive a data set including data about customers only and no business date, it is important that we are able to capture a **date of change** when a change was happening, occurred, registered or took place and came into effect.

Alternative names are ChangeDate, date of effective, EffectiveDate, date of update, UpdateDate, date of use, Applied date, date of creation, SnapshotDate, Timestamp or LogEntryDateTime and maybe date of deletion. In Kimball Design Tip #90 and in Kimball Design Tip #112 date of change is called dimension change date.

Sometimes an operational system has no business date and no transaction date and no event date. I prefer to start a ETL process **before midnight** with Change Data Capture for an operational system with data values to dimensions. Date of change will be set to a current date value at the running time before midnight in staging tables for each dimension. Hereafter column Date of change in a staging table will determine a new value in a ValidFrom column in a type 2 dimension.

Date of change in a staging table has a crucial role for a timeline of a ValidFrom column in a type 2 dimension. I call Date of change for an audit column.

No historical changes, fail and rerun and robust ETL process
Some source systems don't store historical changes, therefore it is important that a type 2 dimension is processed regularly daily to detect changes and add new rows. That way, we can detect changes shortly after they occur, and their validity dates will be accurate.

What will happen in case of a delay to after midnight or a fail or a rerun the next day? A type 2 dimension and a fact can easily display incorrect data!

Of course there is a risk for a ETL process to fail and it will be fixed the day after or a weekend after or holy holiday after together with a rerun. The point is which date or dates will the rerun be using for a new value in a ValidFrom column in a type 2 dimension, see later about a data warehouse clock.

When a new value in a ValidFrom column is a »wrong date«, it is not good for the data quality when a fact does a date range lookup for a dimension key using wrong time period or time span in ValidFrom and ValidTo. A later join of fact and dim tables by the dimension key will not show the right registered values as they were happening, occurred, registered or took place in an operational system!

To implement a robust ETL process to handle reruns it is important to make a **data profiling** of an operating system or a source system for understanding, how data will be delivered and maybe redelivered and hopefully with a date of change or a datetimestamp telling about the data set or the individual rows e.g. date of sale or time of purchase, so we can determine a date or a time for a new value in a ValidFrom column in a type 2 dimension.

When a ETL process runs more than once a day, a ValidFrom column must be able to contain a time, data type of datetime, datetime2(3) or datetime2(7).

Data warehouse clock
The Data Warehouse Framework DWF from Denmark has a »global data warehouse clock« as a datetimestamp stored in a table Meta.Now, which all tasks in a ETL process use to determine a »current time« that is used to fetch data from source systems, and to determine a new value in a ValidFrom column when there is no date of change. Most often the »current time« is the server time and is assigned by a procedure Meta.SetNow without a specific parametervalue. In case of reruns, the »current time« can be turned back to the time of a fail in a ETL process. A ETL process usually runs daily, but for example did it fail last Monday. Today is Friday and the bug has been fixed, tested and deployed to production and we must reestablish correct data in the data warehouse by setting the »current time« in table Meta.Now to an appropriate datetimestamp for Monday, and start a ETL process to load dimensions and facts as it would have happened last Monday, and therefore it will not fetch source data from Tuesday to Friday. The new date in a ValidFrom column will be Monday. The »current time« is set forward one day and the data for Tuesday is

loaded and this process is repeated until data is up-to-date. The framework dimension current view will use »current time« from Meta.Now as a point-in-time, because current dimension values depends of the timeline for a ETL process, e.g.:

```
CREATE VIEW Dim.Customer_Current AS
SELECT d.Customer_key, d.CustomerName, d.City
FROM Dim.Customer d CROSS JOIN Meta.Now n
WHERE n.Now >= ValidFrom AND n.Now < ValidTo
```

HR system example for ValidFrom and ValidTo
A HR system or an Employee management system (EMS) has a time period with business dates for employment start date and employment termination date which is useful for a type 2 dimension ValidFrom and ValidTo.
First time an employee enter the Employee type 2 dimension I will add two rows, where ValidFrom in the second row will represent the employment start date, thus all employees get the same Beginning of time.
For a resigned employee who some years later is rehired I will add an extra row to have no gap in the time span together with a metadata column called IsEmploymentBreak.
Example of content in an Employee dimension, where an employee changes skill and name over time and is resigned and later is rehired.

Employee _key	Employee _bkey	Name	Skill	ValidFrom	ValidTo	IsEmployment Break
293	IW307	Ian Wu	None	1900-01-01	2004-08-01	1
294	IW307	Ian Wu	CIO	2004-08-01	2008-10-01	0
503	IW307	Ian Wu	CFO	2008-10-01	2011-03-01	0
699	IW307	Ian Li	CFO	2011-03-01	2018-01-01	0
875	IW307	Ian Li	None	2018-01-01	2022-01-01	1
993	IW307	Ian Li	CEO	2022-01-01	9999-12-31	0

Columns for Employment start date and Employment termination date can be included the Employee dimension with the same date values as in the HR system.

6.5.3. Type 2 dimension sql Merge implementation

An example of a SalesMart database which consists of a number of tables in different database schemas which is the basis to perform a sql Merge implementation into a type 2 dimension, and it is the same for a type 7 dimension.

This code example including several suffixes and actions:
- _key is a dimension key that is assigned inside the dimension.
- _dkey is a durable key from a Key mapping table that translate a business key.
- _bkey business key because some business uses like it for search criteria.
- _pkey column for a previous dimension key value (parent-child).
- ValidFrom determination by Date of change, a business date from source.
- Early arriving fact as an unknown value marked as inferred member in dim.
- Late arriving dimension marked as inferred member to be corrected in dim.
- Correction of data from a mistyping, typos or misspelling in a source system.
- Deletion of data in a source system as marked deleted member in dimension.

An input table with customer data from a source system that is called CRM.

Input.Customer_CRM

Column name	Description
CustomerId (PK)	Primary key in source, business key in dwh
Name	Name of customer, in dimension CustomerName
Address	Living address of customer, in dimension Address
City	Living city of customer, in dimension City
BusinessDate	When a change was happening in source system
DataStatus	Current, Corrected, Deleted data in source system
RecordSource_audit	Data lineage back to source system, file, table
InsertTime_audit	Audit date and time for inserting row
InsertExecutionLogId_audit	Reference to table Audit.ExecutionLog

A clustered primary key on the business key CustomerId to ensure uniqueness and no duplicates.

Example does not show how data is captured from source system to input table.

A key mapping table to transform and integrate business key values from two source systems with data lineage columns to an artificial auto-generated unique sequence number as a durable key _dkey for later use in a dimension to glue multiple rows together in a timeline for a type 2 dimension.

Keymap.Customer

Column name	Description

Customer_dkey (PK)	Durable key, result of business key transformation
CustomerId_bkey_CRM	Business key from CRM source, not allow null
CustomerNo_bkey_ERP	Business key from ERP source system, allow null
ValidFrom_meta	Timeline metadata
ValidTo_meta	Timeline metadata
InsertTime_audit	Audit date and time for inserting row
UpdateTime_audit	Audit date and time for updating row
InsertExecutionLogId_audit	Reference to table Audit.ExecutionLog
UpdateExecutionLogId_audit	Reference to table Audit.ExecutionLog

A constraint clustered primary key on the durable key Customer_dkey as is an artificial auto-generated unique sequence number or an identity column generated by the database system to uniquely identify each row.

A constraint unique key (unique nonclustered index) for business key CustomerId_bkey_CRM to ensure uniqueness and no duplicates and to speed up to fetch a durable key value from a business key value. ValidFrom_meta will be included. A constraint unique key for business key CustomerNo_bkey_ERP only for not null.

Some data engineers like to hash a business key and save as a binary 64 hashbyte value in column Customer_dkey. I prefer an integer value for _dkey, _key and _pkey.

Example does not show how data is captured from a source system to a key mapping table.

A staging table to shape and suit data to fit a Customer dimension type 2 with audit and metadata columns used from a staging table to merge into a dimension table. Staging table gets data from two input tables Customer_CRM and Customer_ERP.

Staging.Customer

Column name	Description
Customer_dkey (PK)	Durable key from a key mapping table as primary key to merge join with Customer dimension
CustomerId_bkey	Business key from CRM as a business request
CustomerName	Name of customer from input table, not allow null
Address	Address of customer from input table, allow null
City	City of customer from input table, not allow null
BusinessDate	When a change was happening, business date is from input table, allow null for an inferred member
RecordSource_audit	Data lineage back to source systems, input tables
DateOfChange_audit	Copy of BusinessDate for help to ValidFrom and 2006-01-01 is a ETL process default value when BusinessDate is null for an inferred member
IsCorrection_meta	1 for a correction of data of a row in dimension when input table had DataStatus = Corrected or DataStatus = Current for a business key value in dimension or to handle »Late arriving dimension« with IsInferred_meta = 1 to achieve correct values, else 0
IsDeleted_meta	1 for a deletion of data of a row in dimension when input table had DataStatus = Deleted, else 0
IsInferred_meta	1 for an inserting of an unknown row in dimension because there is no descriptive data values to business key from input tables for customer or sales to handle »Early arriving fact«, else 0
InsertTime_audit	Audit date and time for inserting row
UpdateTime_audit	Audit date and time for updating row
InsertExecutionLogId_audit	Reference to table Audit.ExecutionLog
UpdateExecutionLogId_audit	Reference to table Audit.ExecutionLog

A constraint clustered primary key on the durable key Customer_dkey.

A constraint unique key (unique nonclustered index) for business key CustomerId_bkey to ensure uniqueness and no duplicates.

Example does not show how data is captured from input tables to staging table, and how the metadata columns are calculated by a ETL process.

Column DateOfChange_audit gets the value from column BusinessDate that represent when a data change was happening in source system.

In case of no business date I will set column DateOfChange_audit to a current date value and set the ETL process for dimensions to run before midnight.

Dimension.Customer (type 2 or type 7)

Column name	Description
Customer_key (PK)	Dimension key as primary key of the dimension
Customer_dkey	Durable key from key mapping table via staging table
Customer_pkey	Previous dimension key as a reference to a _key
CustomerId_bkey	Business key from CRM as a business user request
CustomerName	Name of customer from staging table, valid by the timeline metadata, not allow null
Address	Address of customer from staging table, valid by the timeline metadata, allow null
City	City of customer from staging table, valid by the timeline metadata, not allow null
BusinessDate	When a change was happening, business date is from staging, allow null for an inferred member
RecordSource_audit	Data lineage back to source systems
DateOfChange_audit	Copy of BusinessDate for help to ValidFrom
ValidFrom_meta	Timeline metadata when values was current truth
ValidTo_meta	Timeline metadata when values was current truth
IsCurrent_meta	1 when row represents current value in a source system when staging table had new descriptive data values for business key value (durable key), else 0 for an historical value from a source system
IsDeleted_meta	1 for deletion of data when row was deleted in source system when staging table had IsDeleted_meta = 1, else 0 for an existing value in a source system
IsInferred_meta	1 when row represents a business key value from a source system with no descriptive data values that become an unknown in dimension when staging table had IsInferred_meta = 1. Later it can be corrected when staging table has IsCorrection_meta = 1, else 0 for a known value in a source system
CorrectionTime_audit	Time for correction of data when row was updated in dimension when staging table had IsCorrection_meta = 1, allow null
InsertTime_audit	Audit date and time for inserting row
UpdateTime_audit	Audit date and time for updating row
InsertExecutionLogId_audit	Reference to table Audit.ExecutionLog
UpdateExecutionLogId_audit	Reference to table Audit.ExecutionLog

A constraint clustered primary key on the dimension key Customer_key as is an artificial auto-generated unique sequence number or an identity column generated by the database system to uniquely identify each row.

A constraint unique key (unique nonclustered index) for Customer_dkey + ValidFrom_meta covering ValidTo_meta and Customer_key to speed up a date range lookup when a staging fact row wants to fetch a dimension key value for a durable key value, alternative for a business key.

A non-unique nonclustered index for Customer_dkey covering IsCurrent_meta to speed up a Current view to fetch a durable key value and values in SQL:
Where IsCurrent_meta = 1.
Alternative a unique index for Customer_dkey with Where IsCurrent_meta = 1.

Inserting a new row into a fact table will date range lookup a durable key value in a dimension table to be between timeline metadata columns ValidFrom and ValidTo to fetch a dimension key value for the new fact row with this criteria using a staging table on the way to load a fact table:

```
stagingfact.Customer_dkey = dim.Customer_dkey AND
stagingfact.PurchaseDate >= dim.ValidFrom_meta AND
stagingfact.PurchaseDate < dim.ValidTo_meta
```

When a source system tells a data warehouse that a business key value has been corrected in source system, the ETL process will overwrite the current row in a type 2 dimension and set a datetimestamp in CorrectionTime_audit and UpdateTime_audit.

When an inferred member gets a real value, the ETL process will overwrite the current row in a type 2 dimension and set IsInferred_meta to 0 and set a datetimestamp in CorrectionTime_audit and UpdateTime_audit.

When a source system tells a data warehouse that a business key value has been deleted in source system, the ETL process will fetch the current row in a dimension

and mark it not-current by set IsCurrent_meta to 0 and update ValidTo_meta, and insert a new row in the dimension and mark it current by set IsCurrent_meta to 1 and mark it deleted by set IsDeleted_meta to 1. A data warehouse will know when a dimension value has been deleted in a source system, because ValidFrom_meta for the mark deleted dimension value row will represent a date for deleting.
No fact row will never refer to a mark deleted row in a dimension.

It is import to know when a column does allow null or does not allow null, because a comparison of a staging table column and a dimension table column must handle a null with a replacing character, or other method e.g. hash difference HashDiff column.

Example of staging table and dimension table with data values
Three staging data day 1 – 3 scenarios for three ETL processes from a staging table with customer data to a dimension table of type 2 with customer data together with three views and their sql code in illustration.

An implementation of a SQL Server stored procedure using a Merge to insert and update rows for a type 2 dimension or a type 7 dimension with durable key is made in illustration, and its include correction of data, mark delete of data in a new row, overwrite an inferred member and using a date of change for ValidFrom determination.

6.6. Metadata column with a prefix or a suffix
A developer like to prefix or suffix all key columns, audit columns, ETL process execution log columns, error description columns and metadata columns to make sure that a source system does not have a column with the same name.

A suffix for audit columns could be **_audit**, for example:
- InsertTime_audit
- UpdateTime_audit
- InsertExecutionLogId_audit
- UpdateExecutionLogId_audit

I recommend to include a data area in the suffix to make columns distinct in case I like to include them in different areas for audit and data lineage purpose e.g.:
- InsertTime_auditIDA
- InsertTime_auditARA
- InsertTime_auditDSA
- InsertTime_auditEDW
- InsertTime_auditDMA
- RecordSource_auditIDA a data lineage back to a source system, a file or to the Usage supporting database with mapping/rule/master data
- RecordSource_auditARA a data lineage back to input data area
- RecordSource_auditDSA a data lineage back to archive area
- RecordSource_auditEDW a data lineage back to data staging area
- RecordSource_auditDMA a data lineage back to enterprise data warehouse

In the DMA data mart area it may be convenient to have InsertTime from then ARA Archive area to audit some delay in the data flow with a timediff.

A suffix for metadata columns could be **_meta**, for example:
- IsCurrent_metaDMA (Flag validity metadata)
- IsDeleted_metaDMA
- IsInferred_metaDMA
- IsCorrected_metaDMA
- ValidFrom_metaDMA (Period validity metadata)
- ValidTo_metaDMA

I recommend to you to have a metadata driven approach to data warehousing.
Audit columns and Metadata columns includes some kind of documentation.
For example, a UserStoryId_meta to Jira and a EnrichmentId_meta to a table row in the Usage database which describes, explains and justifies (reason) a rule of a sql left-join and an enrichment of a derived column in a sql case-when statement. (Extraction and Transformation metadata). I prefer to have mapping data og rules data in Mapping tables and Rules tables in a Usage database instead of having them inside the sql statements.
Why is metadata necessary in a data warehouse (read more):
- First, it acts as the glue that links all parts of the data warehouse.
- Next, it provides information about the contents and structures to the developers and for the operation (DevOps).
- Finally, it opens the doors to the business users and makes the contents recognizable in their terms.

7. Deletion of data in a source system - data deletion
Data is not immortal. Data will be altered, changed, edited, updated and deleted together with deletion of a data row in many source systems.

I like a data warehouse to have a delete metadata column to mark a soft delete:

IsDeleted = 1 when the row represents a deleted row from a source system.
IsDeleted = 0 when the row represents an existing row in a source system.

Column IsDeleted can also be called DeleteFlag, DelFlag, DeletedAtSource, Expired-Flag or InactiveFlag.

To have an IsDeleted column in an archive table and later in a dimension table and in a fact table, the data flow from a source system to Input data area needs some rules to avoid problems, because I have seen data warehouses where many data rows has been marked IsDeleted = 1 and data is still in the source system.

When data capture from a source system is based on incremental load, the source will never tell the data warehouse when a data row is deleted. For example, every month a source system push the latest month data to the data warehouse and therefore older data that has not been changed will not be included in the monthly delivery data e.g. a csv file. In case the data warehouse has implemented a »not in« or »not exists« to mark the Deleted flag, the older data rows will be marked with IsDeleted = 1 even though data still exists in the source system.

Data capture from a source system must be based on full load meaning all data rows are transferred to data warehouse so it can carry out a delta data detection with an appropriate comparison between Input data area and Archive area or dimension/fact to mark IsDeleted = 1 when data really not exists in the source system. The problem with this approach is the amount of data because transferring many million data rows and do comparison can cost a lot of the total ETL process time performance. A vulnerability is if a source system by a mistake only provide half of the data rows, then the data warehouse will think that the other half of data rows has been deleted in the source system and therefore data warehouse will mark many data rows in archive as deleted.

I recommend having a IsDeleted flag in data warehouse only when a source system provides data rows that is marked as Deleted (or D), therefore a source system explicit tells the data warehouse which data rows or information that does not exists anymore and therefore are gone forever. It can be provided with a special file or table but of course it sets requirement for a source system. When a source system has a historical log table as shown in the first chapter, it can be part of the data capture delta load with delta data detection.

8. Archive

An archive is the data area after input data area as described in section 1.7.4. The main purpose for an archive is to capture all changes and remember data that has been changed/modified/updated or removed/deleted in a source system in case they do not storage it themselves. Compliance requirement as »maintaining the chain of custody« of data means that data in an archive cannot be changed instead must insert a new row to hold a revision of data (Kimball page 493).

A popular danish song from 1932 has the title: If you forget, I remember every-thing (in danish Glemmer du så husker jeg alt).

An archive is needed because a data warehouse will seldome include all data from multiple source systems but after a while data warehouse wants to be extended with extra data. Continuous replacement of source systems, rules from public authorities and financial accounting administration is other reasons for an archive. The archive is good for recoverability of a source system or for reload and recovery of a data warehouse.

I have implemented data warehouse without an archive because I included all data from the source systems into historical tables in the data warehouse and I never empty these tables.

I like to think of an archive that it will make it easy for me to see data from multiple source systems at a specific point-in-time.

An archive will have the same grain as a source system and sometimes a data warehouse will have data in a higher grain to reduce query performance. Normally a data warehouse will contain the data that is used in business requirements specification or user story from stakeholders and business users and later extra data can be added from the archive to the data warehouse when needed.

I like to divide the archive into multiple databases where each source system has its own archive database, just like I do it with multiple input data area databases for each source system.

For a ETL process I have to choose between two data flows:

- From Input data area to Archive and from Input data area to Data staging area.
- From Input data area to Archive and from Archive to Data staging area.

In the first data flow the archive is just a storage and will only be used for ad hoc query. In the second data flow the archive must be helpful in loading data to the Data staging area and to Data marts (data warehouse). Archive can be placed on a multiple server environment with cheap hard drive and less memory compared to a data warehouse server.

Lets look at four ways to load data from an archive.

8.1. Full load

The data warehouse will empty all facts tables and all dimension type 1 tables and pull the newest data out of the archive and load data into the data warehouse through a ETL process. Historical dimension like type 2 or type 7 will not be emptied and newest data from the archive will be compared with the dimension and data is merged and inserted. When counting all rows in all tables in the data warehouse data marts is less than 5 million rows, my general rule of thumb is to use full load from archive. When there is business user access to the data through sql statements, it is not good to empty the tables in the middle of the day, therefore full load ETL process is often running once a day, typically in the night, therefore in the afternoon we can't display data from the morning. Instead of empty/truncate a table we can at first load data into a staging table, drop the original table and rename staging table to the original table, or truncate the original table and do a partition switch of staging table into original table.

8.2. Delta load by data compare

The data warehouse will not be emptied and will be loaded with new data or changed data through a ETL process that compare data between archive and data warehouse data marts to find the differences. There will be used techniques such as merge and insert-not-exists, update-join and delete or flag data as soft deleted. When the amount of data is less than 20 million rows is it affordable to compare data specially if a ETL process can run in less than two hours and therefore can be running several times in a day to provide a real-time nature if the source systems support that too.

8.3. Delta load by data batch

When a source system delivery data in a batch (a group of rows, in danish et parti data rækker), the archive needs to have a metadata column like a BatchId for each row in each table to mark all data in the same batch. The data warehouse will pull data for each batch at the time through a managing log in a System supporting database, and data warehouse will compare the data of the batch to find the differences since the latest load.
A batch can also be based on changed Oracle data blocks or SQL Server pages.

8.4. Delta load by data stream

The data warehouse will not be emptied and will be loaded with new data or changed data through a ETL process that use delta data detection by comparing the source input with the target table, and the archive provides audit columns:

- RecordSource is a reference back to the source system e.g. value "Dynamics365.Sales.Product".
- RecordSourceId is a reference back to the data origin primary key in the source system.
- IdaBatchDataCaptureId is a reference to a system table for the input data area.
- IdaInsertTime to indicate when data was inserted into the input data area for measure a delay to when data was inserted into the archive, ArcInsertTime.
- ArcStorage is a reference to a table in a database at an instance at a server, to a data lake and other storage areas.
- ArcBatchDataCaptureId where all rows in all tables will use the same number of identification together with a ArcBatchDataCaptureTime as a received date and time to represents a snapshot of the total amount of data as it look likes in a source system.
- ArcInsertTime (datetime2(7) default sysdatetime(), not unique per row).
- ArcTs (timestamp, rowversion, unique per row).
- ArcRecordId is a unique sequence number per row per table for uniqueness to obtain data lineage and traceability and to facilitate extension of columns from archive to data marts in a data warehouse solution. ArcRecordId can also be used to fetch and pull delta data out of the archive that have not yet been loaded into the data warehouse.
- ArcGlobalId is a unique sequence number per row across all tables in archive.
- ArcRecordState column to tell data is:
  - New e.g. a new customer did not exists before in the archive.
  - Changed e.g. an existing customer has some data changed in the source.

- Removed e.g. a source system provides a delete flag.

ArcRecordState can also be a tiny integer as:
1 = New/Inserted, 2 = Changed/Updated, 3 = Removed/Deleted.

ArcRecordId will get a unique index and query ArcRecordId > latest value will force the index to make sure of a index seek when the data warehouse pulls the delta data from the archive.

I will not recommend the use of metadata columns like ArcInsertTime, Inserted-Date, LastUpdatedDate, LastUpdate, LatestTS or InsertUpdateTimestamp for delta data detection, because a batch of rows will get the same datetimestamp. If not all of data rows is committed before a ETL process pulls out data, some data rows will not be included in the load to the data warehouse, and since they get same datetimestamp, they will not be fetched in the next ETL process. Therefore I prefer using a ArcGlobalId across all tables in archive, but I know some Relational Database Management System RDBMS does not have a global unique sequence number or it will cost performance while inserting into archive tables.

The highest loaded ArcRecordId value for each table in archive will either be stored in a managing log in a System supporting database to be used as »latest value« in the next ETL process, or the highest loaded ArcRecordId value will be found at the beginning of a new ETL process by doing a max query for each table in the data warehouse.

The advantage for a ETL process is there is no need to compare data or to do a lookup to see if data already exists in the data warehouse because the ArcRecord-State flag will be used in the ETL process data flow:

· New will perform a bulk insert of new data into dimensions and facts.
· Changed will perform a type 1 update or a type 2/type 7 merge/update-insert.
· Removed will perform a way to mark data as soft deleted in dimensions and facts.

Hopefully a ETL process can have a fast performance in a first-level data mart with base-level facts, and a delta load can run several times in a day, or when it is finish, it will start again (loading occurs continuously) to keep data in the data warehouse in a near real-time nature if the source systems support that too.

A ETL process from source systems to data marts become a streaming data flow. Of course not all calculations, summary, aggregation, derived data, enrichment and quality check can be based on streaming. With a streaming ETL processing it is a data warehouse task to look at each data row and determine what to do with it and have the responsibility.

Sometimes a source system can provide metadata telling about a data row is new, changed or removed (deleted) which is useful for an ArcRecordState in an archive, else an archive will lookup to see if a data row is new or already exists.

State New is to do a bulk insert load into a target table in a ETL process.

State Changed can be handled as row-by-row updating of target table because one business key can have been updated multiple times in a source system and will be in multiple rows in an archive and maybe it is needed to update multiple times at target to make sure it is matching the source system.

State Removed makes ETL process decide if data is going to be deleted in target table or data will be flagged as soft deleted in a IsDeleted metadata column. To handle deletion of data from a source system to an archive, I recommend the »new row« approach that I used in the example at section 6.3, because the new row will get a new ArcRecordId number value and ArcRecordState becomes Deleted and the new row will contain the newest version of data. On the other side when a source system can provide metadata about data deletion in a column, I think it is not up to the archive to handle it with a IsDeleted metadata column, it must be the data warehouse and data mart to take care of the deletion.

Delta load from a source system to a landing zone area and/or an input data area and to an archive, and delta load from an archive to a data mart is not easy to implement, and it is useful to have a data profiling document to understand a data profile of a source system, e.g. a bank transaction will never be duplicated and will never be updated or deleted because there will instead be a new transaction that is a counterpart, therefore a ArcRecordState metadata column is not needed for an incremental load.

When a message queue is used the order of json files must be secured.

SQL Server has Change Data Capture CDC and Temporal table for auto generated data history which can be used for a source system to transfer changed data to an archive to support incrementally loads change data from the archive tables to the

data warehouse tables. Be aware of table structure changes like a new column, because CDC create a new instance and your ETL process must take care of that.

Instead of a ArcRecordState metadata column there can be metadata columns as IsNewest, IsChanged, IsRemoved (IsDeleted). There can also be a ArcPreviousRecordId that is a reference to the previous data row.

8.5. Example of extraction data from an archive table

An archive can be implemented in a very simple form that look like Kimball type 2 as shown in section 6.2.2 with timeline metadata columns ValidFrom and ValidTo (Effective date and Expiration date) and ArcInsertTime to know when data was added to the archive.

An example of an archive Customer table.

ArcRecord Id	ArcRecord State	UId	CustomerName	ValidFrom	ValidTo
323	New	176	Hans Andersen	1900-01-01	9999-12-31
465	New	359	Joe Watson	1900-01-01	9999-12-31
578	New	421	Marie Beaulieur	1900-01-01	2007-11-16
653	Changed	421	Marie Lesauvage	2007-11-16	2014-07-15
746	Changed	421	Marie Sainte	2014-07-15	2017-02-07
975	Removed	421	Marie Sainte	2017-02-07	9999-12-31

Ad hoc query to extract the newest loaded data from an archive table:

```
SELECT CustomerName
FROM arc.Customer
WHERE ValidTo = '9999-12-31'
```

Example of traceability in a type 1 dimension for Customer.

Customer_key	CustomeId_bkey	CustomerName	ArcRecordId
1	176	Hans Andersen	323
2	359	Joe Watson	465
3	421	Marie Sainte	578, 653, 746, 975

Marie Sainte has changed surname multiple time and she is still part of the dimension because her old purchases is in the fact.

I recommend to use of a business date or a date of change from an operational system or a source system to determine a new value in a ValidFrom column to avoid the problem that loading into an archive is done after midnight, e.g. a deposit into a bank account is happening December 31 must have ValidFrom at same date to show the person's fortune in the actual year according to a deposit date column.

An example of an archive hotel room table without timeline metadata columns ValidFrom and ValidTo, instead an actual time period or time span by the columns FromDate and ToDate, therefore with no overlap for a hotel room.

ArcRecord Id	Room Number	Status	FromDate	ToDate	ArcInsertTime
323	485	Booked	2017-04-03	2017-04-05	2017-01-23
465	104	Booked	2017-04-07	2017-04-14	2017-01-24
578	485	Cancelled	2017-04-03	2017-04-05	2017-01-24
601	104	Adding	2017-04-07	2017-04-15	2017-01-28
606	104	Confirm	2017-04-07	2017-04-15	2017-01-28
634	104	Paid	2017-04-07	2017-04-15	2017-01-30
4994	104	Checkin	2017-04-07	2017-04-15	2017-04-07
4999	104	Occupied	2017-04-07	2017-04-15	2017-04-07
5393	104	Checkout	2017-04-07	2017-04-15	2017-04-15
5416	104	Cleaning	2017-04-15	2017-04-15	2017-04-15
5478	104	Available	2017-04-16	9999-12-31	2017-04-16

Row 578 is a counterpart data row. If the data warehouse has been down from 2017-01-23 to 2017-01-25 and it only fetchs the newest row from the archive, room number 485 could lead to a misunderstanding because it is cancelled without beeing booked first. Therefore we need all the rows from the archive to pass through the data warehouse in the order of ArcRecordId in a streaming oriented ETL process, streaming load compared to a batch load with many rows at the same time.

If you like timeline metadata columns ValidFrom and ValidTo then make a view:

```
SELECT RoomNumber, Status,
       ValidFrom = FromDate,
       ValidTo   = LEAD(FromDate, 1, DATEFROMPARTS(9999, 12, 31))
                   OVER(PARTITION BY RoomNumber
```

```
                    ORDER BY FromDate, ArcRecordId)
FROM arc.HotelRoom
```

If you like room 485 first row with ValidFrom 1900-01-01 then:

```
ValidFrom = CASE WHEN ValidFrom = MIN(FromDate) OVER(PARTITION BY RoomNumber)
                 THEN DATEFROMPARTS(1900, 1, 1) ELSE FromDate END
```

When you like to fetch a room e.g. 104 at April 1, 2017, you can make a query, a stored procedure or a table-valued function with parameters for room and date:

```
WITH as_was (ArcRecordId) AS
(
 SELECT MAX(ArcRecordId) AS MaxArcRecordId
 FROM arc.HotelRoom
 WHERE RoomNumber = 104 AND ArcInsertTime <= '2017-04-01'
)
SELECT *
FROM arc.HotelRoom
WHERE ArcRecordId = (SELECT MaxArcRecordId FROM as_was)
```

ArcRecord Id	Room Number	Status	FromDate	ToDate	ArcInsertTime
634	104	Paid	2017-04-07	2017-04-15	2017-01-30

In case a source system delivers a sequential id e.g. a RevisionId or a EventId it is better to use it in the reading order from the archive. There could also be a EventType telling the type of change as new, changed or removed. Sometimes a source system deliver a batch of rows that contain duplicate rows e.g. an OrderNumber in three rows with different RevisionNumber and when an archive has a metadata column ArcIsNewest, the archive has to determine which row is the best within the duplicates. The batch can look like this.

RevisionNumber	OrderNumber	OrderDate	OrderAmount
467	90484	2018-09-24	350
481	90484	2018-09-24	356
497	90484	2018-09-24	359

I call OrderNumber for business key because it is the column that the source system and the archive can use to handshake and exchange data row with.

After adding the batch to the archive it will look like this and calculate ArcIsNewest.

ArcRecord Id	ArcInsert Time	ArcIs Newest	Revision Number	Order Number	OrderDate	Order Amount
75736	2018-09-25	0	467	90484	2018-09-24	350
75737	2018-09-25	0	481	90484	2018-09-24	356
75738	2018-09-25	1	497	90484	2018-09-24	359

```
ArcIsNewest = CASE WHEN RevisionNumber = MAX(RevisionNumber)
                                         OVER(PARTITION BY OrderNumber)
                   THEN 1 ELSE 0 END
```

In a later batch two more revision is coming.

RevisionNumber	OrderNumber	OrderDate	OrderAmount
501	90484	2018-09-24	357
507	90484	2018-09-24	358

After adding the batch to the archive it will look like this and calculate ArcIsNewest.

ArcRecord Id	ArcInsert Time	ArcIs Newest	Revision Number	Order Number	OrderDate	Order Amount
75736	2018-09-25	0	467	90484	2018-09-24	350
75737	2018-09-25	0	481	90484	2018-09-24	356
75738	2018-09-25	0	497	90484	2018-09-24	359
76911	2018-09-25	0	501	90484	2018-09-24	357
76912	2018-09-25	1	507	90484	2018-09-24	358

A data warehouse will use OrderNumber to update the order fact row with the row from the archive with ArcIsNewest = 1.

Metadata column ArcIsNewest can work together with a meta column ArcIsDeleted to tell that the value is no longer available in the source system. An example with an order that was made at time 2:10 pm but deleted 15 minutes later and later is reappearing at 3:05 pm and at the end at 3:09 pm is deleted for good, therefore the newest row in the archive is flagged as a deleted row (soft delete).

ArcRecord	ArcInsert	ArcIs	ArcIs	Order	OrderDate	Order

Id	Time	Newest	Deleted	No		Amount
75736	2018-09-25 2:10	0	0	90484	2018-09-24	350
75737	2018-09-25 2:25	0	1	90484	2018-09-24	356
75738	2018-09-25 3:05	0	0	90484	2018-09-24	359
76911	2018-09-25 3:06	0	0	90484	2018-09-24	357
76912	2018-09-25 3:09	1	1	90484	2018-09-24	358

To extract data from the above archive table by **full load** to a data warehouse use criteria: WHERE ArcIsNewest = 1 AND ArcIsDeleted = 0.

To extract data from the above archive table by **incremental load** to a data warehouse use criteria: WHERE ArcIsNewest = 1, because a ETL process needs to handle the flagged deleted row from the archive to the data warehouse to do a deletion of the fact row or to flag it as deleted for a soft deleting. More about incremental load and delta load in next section.

Please notice, that I am using the term »newest« for the latest received data in the archive, and I don't using the term »current« in an archive, because I do find it confusing if I have two meta columns saying: ArcIsCurrent = 1 AND ArcIsDeleted = 1 bind to a transaction row that will become a fact row in the data warehouse. For me it is only in a dimension that a row can be both current and deleted, because it is current for the old fact rows and it is flagged deleted so a future new fact row can not refer to that row in the dimension which a ETL process will handle.

For me a metadata column ArcIsNewest flag is not always right to have in an archive, maybe the data warehouse did not fetch some data row before they changed flag from 1 to 0 and missing data did not reach the data warehouse, other times it is fine, therefore a business requirements specification or a user story is important for the archive too.

In a Data vault modeling I have a satellite table for names of employees: SatEmployeeName(EmployeeHashKey, Name, Gender, LoadDate, DeletedDate), and I fetch the newest row for each employee using a windowing function in sql:

```
SELECT EmployeeHashKey
    ,Name
    ,Gender
    ,LoadDate
    ,DeletedDate
FROM
(
 SELECT EmployeeHashKey
     ,Name
     ,Gender
     ,LoadDate
     ,DeletedDate
     ,ROW_NUMBER() OVER(PARTITION BY EmployeeHashKey ORDER BY LoadDate DESC)
     AS RowNo
 FROM SatEmployeeName
) t
WHERE RowNo = 1
```

8.6. Example of extraction data from archive tables

An **incremental load** combines two archive tables into a persistent staging area PSA table. A joined row will have two columns of ArcInsertTime from the two archive tables, and the greatest of them will become the new LatestArcInsertTime that will be stored in a psa.Orders table, to be the previous LatestArcInsertTime for the forthcoming incremental load.
For example in sql with a left join, if there is no related product and if it arrives tomorrow, the incremental load will not fetch it unless the related order is also updated with a new time stamp in column ArcInsertTime. See the use of full join in next paragraph:

```
INSERT INTO psa.Orders (ReferenceNo, ArcRecordId_Orders, OrderNumber, OrderDate, Quantity,
                        ArcRecordId_Product, ProductNumber, UnitPrice, LatestArcInsertTime)
SELECT
  o.ReferenceNo, o.ArcRecordId_Orders, o.OrderNumber, o.OrderDate, o.Quantity,
  p.ArcRecordId_Product, p.ProductNumber, p.UnitPrice,
  GREATEST(o.ArcInsertTime, p.ArcInsertTime) AS LatestArcInsertTime
FROM arc.Orders o
      LEFT OUTER JOIN arc.OrdersProduct p ON p.ReferenceNo = o.ReferenceNo
WHERE GREATEST(o.ArcInsertTime, p.ArcInsertTime) >
        (SELECT ISNULL(MAX(LatestArcInsertTime), '0001-01-01 00:00:00') FROM psa.Orders)
```

Incremental loading assumes that the archive tables are at rest, meaning that there is no other process doing inserting, updating, or deleting rows in the archive tables while the incremental load is running.

A database snapshot is a read-only, static view of a database similar to materialized view, e.g. for archive tables that can be made just after a load to the archive tables and the incremental load is fetching data from the database snapshot. Therefore

the next load to the archive tables can be running at the same time as the incremental load because it is based on the a database snapshot.

A **delta load** combines three archive tables into a persistent staging area PSA table, where I delete exiting rows instead of update them before I insert rows (upsert), when a new version (edition or state) of data for an order has arrived to the archive.

An order source system sends data to three archive tables, that insert data as rows together with values in metadata columns ArcRecordId and ArcInsertTime.
The order source system can change data over time for OrderStatus, Quantity, ProductNumber and UnitPrice due to a correction of data in the source system.
A change of data is called a new version of data.
The archive tables are a changelog with only inserted rows and no updated rows, therefore an order has multiple versions of data in multiple rows.
The current version of data has the latest time value in ArcInsertTime.

An order has up to three statuses: Ordered, Shipped, Delivered.
An order has a ReferenceNo value which exists across the three archive tables where the ReferenceNo column binds the rows together.

The three archive tables will be merged into a psa.Orders table based in two rules:
- When an order changes status, insert a new row with the status in psa.Orders.
- When an order changes data, update the row with the data in psa.Orders.

A psa.Orders table row represents one order with one status with the current version of data.

When there is a change in a data value column, e.g. OrderStatus, Quantity, UnitPrice in the order source system, it will become a new row in an archive table with the same value in column ReferenceNo as the previous rows.
An order with three statuses is stored in three rows in psa.Orders with the same value in column ReferenceNo.
ReferenceNo is a business key for a row that has different columns that can change data values over time, which is handled by the archive tables by adding a new row.

The psa.Orders table is the result of merging rows from the three archive tables, and it has a composite unique nonclustered index of ReferenceNo and OrderStatus, and a clustered primary key of PsaRecordId_Orders as a unique sequence number per row, where a row contains the current version or the latest version of data from the archive tables.

Example data in three archive tables and result of delta load to psa.Orders table

The psa.Orders table contains a LatestArcInsertTime column that is used for a delta data detection with a default datetime when the PSA table is empty before an initial load or a full load. It is an alternative to having a managing log in a system supporting database to save a value of the »latest value« for incremental load or delta load with delta data detection.

The three archive tables are full outer joined to handle when a ReferenceNo does not exists in all tables at the extract time. A joined row will have three columns of ArcInsertTime from the three archive tables, and the greatest of them will become the new LatestArcInsertTime that will be stored in the psa.Orders table, to be the previous LatestArcInsertTime for the forthcoming delta load. For example in sql:

```
WHERE LatestArcInsertTime > (SELECT ISNULL(MAX(LatestArcInsertTime), '0001-01-01 00:00:00')
                             FROM psa.Orders)
```

Only the recent joined rows from the three archive tables will be extract as a delta load to the PSA table, and the many old rows in the archive tables are untouched with older values in the ArcInsertTime column.

In the sql code below, a stored procedure in T-SQL at SQL Server, to handle delta load, there is a removing duplicates of rows with same value in ReferenceNo and OrderStatus to fetch the row with the greatest LatestArcInsertTime value, because a full outer join can end up with duplicate values when the archive tables has received multiple rows with the same value in ReferenceNo and OrderStatus before a ETL process has been running and update (delete/insert) the psa.Orders table.
Errors and crashes must be taken into account when a ETL process has not been running for a few days.
LoadType use Incremental as default value, else call the stored procedure with Full.
For example in sql:

```
CREATE PROCEDURE [dbo].[Orders_Insert]
  @LoadType varchar(20) = 'Incremental' AS
BEGIN
SET NOCOUNT ON
```

```
IF @LoadType = 'Full'
  TRUNCATE TABLE psa.Orders -- an empty psa table will use datetimestamp 0001-01-01 00:00:00

DROP TABLE IF EXISTS #DeltaOrders

-- Extract new version of ReferenceNo rows and new ReferenceNo rows since latest run of ETL process
;WITH dataset AS
(
SELECT
  COALESCE(o.ReferenceNo, p.ReferenceNo, s.ReferenceNo) AS ReferenceNo,
  o.ArcRecordId_Orders, o.OrderNumber, o.OrderDate, o.Quantity,
  p.ArcRecordId_Product, p.ProductNumber, p.UnitPrice,
  s.ArcRecordId_Status, s.OrderStatus,
  ISNULL(GREATEST(o.ArcInsertTime, p.ArcInsertTime, s.ArcInsertTime), '9999-12-31 23:59:59') AS
  LatestArcInsertTime
FROM arc.Orders o  -- join all rows in archive without no guarantee data exists in all tables at same time
       FULL OUTER JOIN arc.OrdersProduct p ON p.ReferenceNo = o.ReferenceNo
       FULL OUTER JOIN arc.OrdersStatus s ON s.ReferenceNo = o.ReferenceNo OR
                                             s.ReferenceNo = p.ReferenceNo
), withoutDuplicates AS
(
SELECT *
FROM
(
 SELECT *,
   RowNo = ROW_NUMBER() OVER (PARTITION BY ReferenceNo, OrderStatus
                              ORDER BY LatestArcInsertTime DESC, ArcRecordId_Orders DESC,
                              ArcRecordId_Product DESC, ArcRecordId_Status DESC)
                    -- to handle same value in LatestArcInsertTime in several rows, duplicate values
 FROM dataset
) t
WHERE RowNo = 1
)
SELECT *
INTO #DeltaOrders
FROM withoutDuplicates  -- fetch only rows with a time greatest than the fetched latest time
WHERE LatestArcInsertTime > (SELECT ISNULL(MAX(LatestArcInsertTime), '0001-01-01 00:00:00')
                             FROM psa.Orders)

SELECT @@ROWCOUNT AS NumberOfRows -- can be inserted into an audit log table

BEGIN TRY
 BEGIN TRANSACTION

 -- Delete in psa table existing ReferenceNo+OrderStatus rows, because a new version exists in archive
 DELETE psa
 FROM psa.Orders psa
       INNER JOIN #DeltaOrders delta ON delta.ReferenceNo = psa.ReferenceNo AND
                                        delta.OrderStatus =  ISNULL(psa.OrderStatus, delta.OrderStatus)

 -- Insert into psa table new version of ReferenceNo+OrderStatus rows and new ReferenceNo rows
INSERT INTO psa.Orders (ReferenceNo, ArcRecordId_Orders, OrderNumber, OrderDate, Quantity,
 ArcRecordId_Product, ProductNumber, UnitPrice, ArcRecordId_Status,OrderStatus,LatestArcInsertTime)
SELECT ReferenceNo, ArcRecordId_Orders, OrderNumber, OrderDate, Quantity,
 ArcRecordId_Product, ProductNumber, UnitPrice, ArcRecordId_Status, OrderStatus, LatestArcInsertTime
FROM #DeltaOrders

 COMMIT TRANSACTION
END TRY
BEGIN CATCH
 ROLLBACK TRANSACTION
END CATCH
END
```

psa.Orders table can be staging data for the Transactional fact and Accumulating snapshot fact showed in section 3.3. Types of fact tables in paragraph Accumulating snapshot fact using delta load if I had included column LatestArcInsertTime in the fact tables.

For delta data detection I have chosen an ArcInsertTime instead of an ArcGlobalId, because most archive tables I've seen have used a datetimestamp. As times goes by a datetimestamp works the same way as a GlobalId sequence number but of course several rows will get the same datetimestamp value.

Embedded SQL example looking three days back in time because no latest column

```
select *
from {{ref('events')}}
  {% if is_incremental() %}
    where event_timestamp >= (
      select dateadd(day, -3, max(event_timestamp)::date) from {{this}}
      )
  {% endif %}
```

This approach establishes a »window« of three days recent data to always re-transform every time the ETL process runs. Essentially, late arriving facts have three days to arrive, after which point they are no longer eligible to be incorporated into our metrics (until, at least, we do a full load). This strategy is just what it seems: close enough. If you require perfect accuracy on every transformation, this isn't a good strategy.

9. Miscellaneous

Agile with overview and consideration is good and creates agility (in danish agile med overblik og omtanke er godt og skaber smidighed).

9.1. User story

A user story is a story told by a business user specifying how the system is supposed to work. It is a popular way of writing lightweigth product requirements with a standard form of: »As a <user role>, I want to <action> to achieve <business value>.« E.g. »As an upcoming passenger, I would like to be able to cancel my flight ticket reservation.« An extra wish is that a premium member can cancel the same day without a fee and for others there is a charged of 10% for the cancellation.

A user story includes definitions, clarifications, calculation formulas and acceptance criteria for how to reach the goal when the acceptance criteria has been met, e.g. scenarios for cancellations as »cancel within 24 hours or 2 hours, with mixed customer types, cancel a cancelled reservation.« A user story also includes a description of the benefits to business users and success factors for the business.

A epic is a collection of user stories that make up a whole.

For a user story do refinement to break down it into tasks (specific activities), and a task can be divided into sub-tasks that can be performed and set a time estimate for each task/activity. A user story gets story points. Time is important for sprint planning of the next sprints.

Remember acceptance criteria for each activity in the user story or for the whole user story.

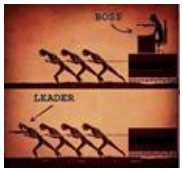
After a sprint is done, please hold a retrospective (in danish set i bakspejlet) meeting for the team to reflect on what happened in the sprint and to identify actions for improvement going forward.

9.2. Agile development

In agile development a team makes new product or updated a product in short increments called sprints. With innovation occurring in rapid intervals, the team can continuously reassess its priorities and more easily adapt to evolving requirements. This type of responsiveness is impossible using a waterfall project management methodology which locks a team into a long development cycle with one big-bang deliverable at the end.

Good points to an agenda for the agile daily scrum standup meeting:

- How is it going? Hvad har vi gang i?
- What is the outgoing? Hvilke ting er færdig?
- What gets in the way? Er der forhindring?



For what gets in the way, a user story is blocked by an impediment which is anything that is slowing down a team.

Example of steps for a task in a scrum board:

- Backlog with tasks with an overall description.
- Breakdown a task to subtasks for a closer analysis and a User story.
- Design with categories for Ready, Work in Progress and Done.
- Development categories for Ready, Work in Progress and Done.
- User acceptance testing categories for Ready, Work in Progress and Done.
- Ready for release and to deploy in production and follow up tests.
- Done Done to keep a Changelog.

Please read more about agile manifesto and agile software development methodology that can lend itself to LEAN principles. Agile are usually a small wheel in a much larger waterfall machine, for it to work, a more system based approach is needed to ensure, that the rest of the organisation can keep up. For me, agile must incorporates some waterfall features (increment and release) to work better. I don't see waterfall and agile as polarities as opposed to stages in the software development. Software development has already moved from working sequentially to work in circular and iteratively. Agilization defines the process of transforming enterprise behaviour from one form to another that meets the competitive imperative of organizations of the 21st century to always fit their most profitable markets. The agile way of working gives a faster delivery of value to the business community, because it has a two weeks deliverable sprint. Google wants to ensure that the best thing that comes out is quality, not how fast something goes out. Google has not

fully adopted one approach for the entire company - it neither uses agile, not waterfall, instead it adopts agile practices to have some project processes, ref.

Kimball page 405: »We believe most situations warrant the bundling of multiple agile deliverables into a more full-function release before being broadly deployed to the general business community.«
Different view of agile and Agile BI the death of Waterfall BI projects?

The Scaled Agile Framework® (SAFe®) is a set of organization and workflow patterns for implementing agile practices at enterprise scale. The framework is a body of knowledge that includes structured guidance on roles and responsibilities, how to plan and manage the work, and values to uphold. SAFe has four models: Essential, Large Solution, Portfolio, and Full. Each uses a fit for purpose hierarchy of items. SAFe portfolio has a four tiered system: Scaled Agile Framework value stream → portfolio (epics) → program (features) → team (user stories and tasks for Scrum practice and items for Kanban practice).

PI is Program Increment with a number e.g. PI24 is a specific period in an Agile Release Train during which teams deliver working software value to run in production. Program Increment Planning or PI planning or PIP. Read more.

Organizational hierarchy for a SAFe EPIC:

- Capability at business level to be solved in one or several PI periods
- Feature a business need to be solved in a PI period
- User story a task to do for a feature to be solved in a Sprint period
- Sub task up to a developer to divide a user story into sub tasks

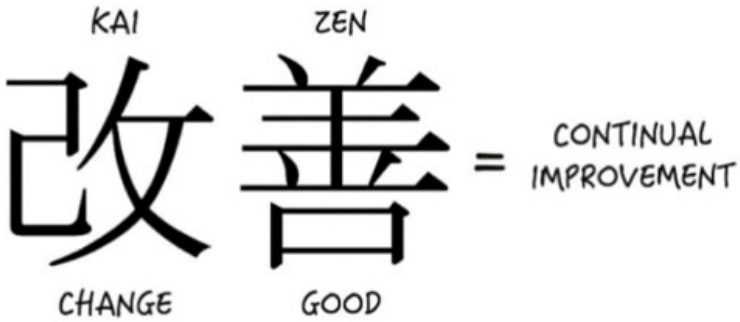
Feature is in Jira called a Epic link.
Capability covers the ability of possibilities to be able to do.

Daily Stand Up DSU agenda:

- What did you do yesterday from tasks on the sprint board?
- What tasks are you going to do today on the sprint board?
- Is there anything preventing progress on the tasks?
- Other important knowledge/information - and/or mutual agreements?

Jira out of the box has a two tiered system epic and user story. I have seen articles and third party solutions for creating the SAFe tiers in the portfolio model. You need to decide on your SAFe model, then find a way to implement that model in Jira. By the way, SAFe epics can be business objectives or architectural enablers. Maybe a Jira issue attribute could be used for this distinction. Azure DevOps is also on the way. Of course each tool like to use their own terms.

Agile is an iterative methodology used for developing a product that focuses on the continuous delivery of tasks assigned. SAFe, on the other hand, is an agile framework for an enterprise which is not limited to smaller teams and guides enterprises in scaling lean and agile practices. Continuous improvement is about making small changes every day or week to reach the long-term goals because it is focusing on the process and not only at the end result. Kaizen is from Japanese wisdom.



My approach and advise to everyone 💎

9.3. Git branching for data warehouse using Azure DevOps

Git is not an acronym but a de facto standard for source code version control system to keep track of all previous versions. Git allows a team of developers to work in parallel development and editing same files and merge files together using branches. The point of Git is, don't move a file instead merge a file into the main file.

It can be a GitHub or an Azure DevOps project containing a repository or several repositories for solutions for backend with database project and ETL project, and for frontend with OLAP cubes and reports. A repository (repos) is a shared folder on a remote server and each developer has its own local repository to stage, commit, sync, push changed files into the server repository where files will be merge and another developer can pull the merged version to a local repository. TFS calls it checkout from the server, checkin to the server and get latest from the server.

Production source code is placed in a default branch called main, mainline, trunk or master but deprecated for unnecessary references to slavery.

Development happens in a feature branch and will be integrated in an integration branch (or develop branch) together with other developers feature branches.

The integration branch is where we bring multiple feature branches together for testing in a test data warehouse. The purpose of the integration branch is to determine whether new features work not only on their own, but also in combination with other new features and for doing full business user acceptance test and QA testing.

· Local feature branch use a development server and a developer runs tests.
· Features is a folder at remote server with all feature branches over time.
· Remote feature branch is to prepare a merge into an integration branch and deployment to test environment.
· Integration branch merge features and use a test server together with few users.
· Local release-version branch use a development server to create a new release.
· Releases is a folder at remote server with all release-version branches over time.
· Remote release-version branch is to prepare a merge into a main branch and deployment to production environment.
· Main branch merge releases and use a production server with access for all users.

Here is my suggestion of branches where I use a local feature branch for all new features including bugfixes and hotfixes in a development environment:

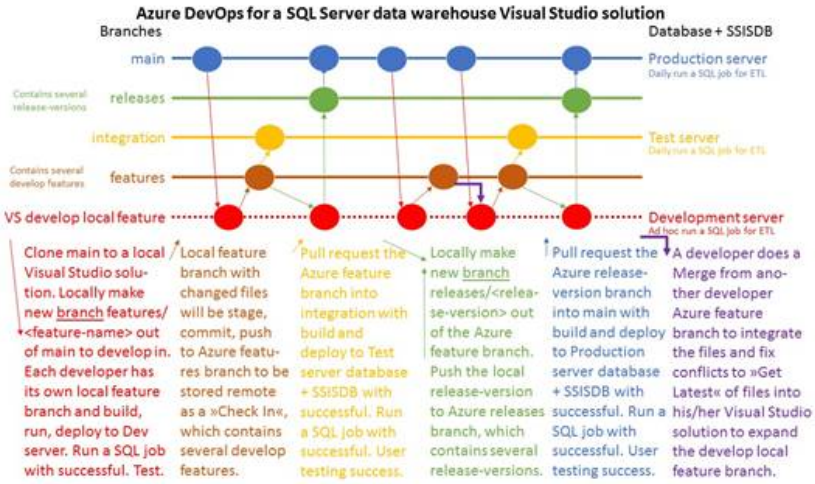
1. Clone a repository, which copy main branch to local editor/IDE.
2. Make a new local feature branch based at main branch and do development.
3. Manuelly build and deploy the local feature branch to development server.
4. Make many tests in the development environment until a successful test result.
5. Push the local feature branch to a remote server features folder.
6. Do a merge from origin/integration because integrate happens in a local solution to avoid a abandon message »conflict prevents automatic merging«.
   Resolve any merge conflicts.
   Again do a push of the local feature branch to a remote server.
7. Pull request for let the feature branch into integration branch.
8. Approve the pull request feature branch by another developer.
9. Merge the pull request feature branch into integration branch.

10. Build, release and deploy it to the test environment.
11. Make many tests in the test environment until a successful test result.
12. Make a new local release-version branch based at the feature branch.
13. Push the local release-version branch to a remote server releases folder.
14. Do a merge from origin/main because integrate happens in a local solution.
    Resolve any merge conflicts.
    Again do a push of the local release-version branch to a remote server.
15. Pull request for let the release-version branch into main branch.
16. Approve the pull request release-version branch by another developer.
17. Merge the pull request release-version branch into main branch.
18. Build, release and deploy it to the production environment.
19. Make many tests in the production environment until a successful test result.

Pull request means you want to pull the content of a feature branch or a release-version branch into the integration branch or the main branch and merge them.

(Pull request in danish at bede om en anmodning om at flette kode ind i en branch integration eller main, som skal godkendes af anden udvikler før det kan ske).

Git flow



When the integration branch is clean and all its parts is tested successfully and everything in integration is ready to go into production, we can make a new local release-version branch based on the integration branch. Hereafter a merge from origin/main because integrate happens in a local solution, and a pull request the release-version branch into main to merge, build, release and deploy it to the production environment.

Git branches are effectively a pointer to a snapshot of changes.

An Azure DevOps project contains one repository or many repositories, and a pull request is into a repository and into integration or main branch of the repository. Pipelines of build and release are at project level with one build pipeline for both test and production, one test release pipeline and one production release pipeline. A build pipeline refer to a repository and has triggers to branches, therefore add the name of the repository as a prefix to the name of the build pipeline. A release pipeline refer to a build pipeline and to a branch, and with two branches integration and main, there will be two release pipelines for test and for production, therefore add the name of the repository and the branch as a prefix to the name of the two release pipelines. Remember that a branch policy must be added to a build pipeline. Please read more about Git.

In DOS stay in folder repos and clone a repos from github:
git clone https://your-token@github.xyz.dk/abc/repos_name.git

Or stay in a folder repos_name:
git status
Switch to branch develop:
git checkout develop
Fetch new/changed files from remote server GitHub to the local repos branch:
git pull

Create a new feature branch called feature_name:
git checkout -b feature/feature_name
Switched to a new branch 'feature/feature_name'
Go to a feature branch:
get checkout feature/feature_name
Merger develop into feature branch:

git merge develop

Create a new file or change a file in the feature branch feature_name in a subfolder
C:\Repos\repos_name\database_table\Customer.sql

List of created or changed files in the feature branch feature_name:
git status
On branch feature/feature_name
Untracked files:
  database_table/Customer.sql

Add the .sql file:
git add C:\Repos\repos_name\database_table\Customer.sql

List of added files:
git status
On branch feature/feature_name
Changes to be committed:
  new file: C:\Repos\repos_name\database_table\Customer.sql

Commit a added file:
git commit -m "sql script for new Customer table"

See a status saying nothing to commit because all files has been added:
git status

Transfer the local repos feature branch feature_name to remote server GitHub:
git push --set-upstream origin feature/feature_name

In browser create a pull request for 'feature/feature_name' at GitHub to merge into a branch:
Click Compare & pull request.
Click Create pull request.
In browser: https://github.xyz.dk/abc/repos_name/pull/292 ready to be approved.
After approve do the merge into the branch at remote GitHub server:
Click Squash and merge.
Ready to deploy to test or production.

9.4. A little bit of other things

Many issues to consider in feasibility a data warehouse.

Language and terms between people is a keystone for collaboration on designing and using a good data warehouse and a BI solution.

In this time of age means »now, at the present time«.

The time can be displayed using both the 24-hour format (0 - 24) or the 12-hour format (1 - 12 am/pm).

Greenwich Mean Time (GMT) is a time zone officially used in United Kingdom, the Republic of Ireland and Portugal and some African countries.

Central European Time (CET) is a time zone officially used in Continental Europe and is 1 hour ahead of Greenwich Mean Time.

Universal Time Coordinated (UTC) is a time standard that is the basis for civil time and time zones worldwide. This means that no country or territory officially uses UTC as a local time. Letter Z stands for the Zero time zone meaning UTC (an offset of zero hour-minutes-seconds-milliseconds) as in 2021-05-26T10:17:38.549Z.
DateTime.UtcNow.ToString("yyyy-MM-ddThh:mm:ss.fffZ").

Daylight Saving Time (DST) do not change GMT or UTC. However, some of the countries that use GMT switch to different time zones during their DST period.
When Denmark has summer time, DST, a local time 17:45 is UTC 15:45.
When Denmark has winter time, normal time, a local time 17:45 is UTC 16:45.

A real-time data warehouse needs to use UTC time to avoid DST, because when there goes from summer time to winter time, the same o'clock happens twice like 2.30 am. Suffix data warehouse columns with _UTC, e.g.:
IdaInsertTime_UTC, ArcInsertTime_UTC, EffectiveDate_UTC, ExpirationDate_UTC, ValidFrom_UTC, ValidTo_UTC, InsertTime_UTC, UpdateTime_UTC, LoadDate_UTC and DeletedDate_UTC together with the default value GETUTCDATE().

Before making a date range lookup, make sure to calculate and to store columns of date from a source system as UTC, e.g.:
TransactionDateTime_UTC = TransactionDateTime AT TIME ZONE
'Central European Standard Time' AT TIME ZONE 'UTC'

Be aware that an ArcInsertTime_UTC says 2010-08-16 23:51:05 and the source data TransactionDateTime says 2010-08-17 00:49:36 because it has its own time zone. TransactionDateTime_UTC says 2010-08-16 23:49:36.

Other issues for a real-time data warehouse. Many time zones.

The term idempotent or idempotency is used to describe an operation that produces the same result no matter how many times it is performed.
The ultimate sophistication is simplification. Emoji.
CamelCase or snake_case or both CustomerId_bkey.
There are two ways to write error-free programs; only the third one works.
When all else fails, read the instructions.
With a strong framework, you can’t make as many mistakes, but you are also less flexible.
A null string is different than an empty string "", because a null string doesn’t point to anything in memory.
The best number is 73, because 73 is the 21st prime number and its mirror 37 is the 12th prime number and its mirror of 21. In binary 73 becomes 1001001 a palindrome same backwards. In morse code 73 becomes — —••• •••— — also a palindrome and it is an old telegraph code that means »best regards«. 73 is the number of books in the Catholic Bible.
The number 42 is the answer to the »ultimate question of life, the universe, and everything« a joke in Douglas Adams's 1979 novel, The Hitchhiker's Guide to the Galaxy.
The mind is too important to waste.
Twins is either Identical (have the same DNA) or Fraternal (two separate fertilized eggs). However identical twins may not look exactly identical to one another because of environmental factors such as womb position and life experiences after being born, so there will always be a unique characteristic of each twin.

Abbreviation is a shortened form of a word or phrase like Dr. for Doctor and Inc. for Incorporation.

Acronym is a word formed from the first letter of every word like RAM for Random Access Memory. Every acronym is a type of abbreviation but not the other way round. DWH is an abbreviation and not an acronym.
Americans say the abbreviation SEQUEL but it was later changed to acronym SQL (Structured Query Language) because SEQUEL was a trademark of the UK-based Hawker Siddeley aircraft company.

Synonym example:
- is occupied at the moment
- is tied up at the moment
- is busy right now
- is not free right now
- is not available
- is unavailable

A data fly ticket and a visual fly ticket.

The Model-View-Controller (MVC) pattern separates the modeling of the domain, the presentation, and the actions based on user input into three separate classes:

- The model manages the behavior and data of the application domain, responds to requests for information about its state (usually from the view), and responds to instructions to change state (usually from the controller).
- The view manages the display of information.
- The controller interprets the mouse and keyboard inputs from the user, informing the model and/or the view to change as appropriate.

A Class with attributes and methods is a template and is not alive, but it contains the blueprint for how something would behave when that something is alive.
An Object is an instance of a class when you create a living thing based on that template. New keyword creates an object in memory and is calling a special class method called a Constructor or a Class_Initialize to assign the attributes with a start value at the time of an object creation (Create, Load, Begin, Start, Setup). At the time of object destruction by a garbage collector an object is calling a special class method called a Destructor or a Class_Terminate (Destruct, Unload, End, Stop, Unset, Finalize, Cease, Clean up, Clear, Close, Destroy, Dispose, Final, Finish, Kill, Release, Remove, Reset).
Inside a method the keyword this and Me is referring to a specific object itself.
What are the three principles of object oriented programming OOP:

- Encapsulation refers to the bundling of data with the methods that operate on that data with inside protection.
- Inheritance of classes in hierarchy.
- Polymorphism is the ability of an object to take on many forms like a parent class reference is used to refer to a child class object.

An interface is the what. An implementation is the how. An interface declares what an object will do. Think of it as a contract, a set of promises. An object that implements an interface must agree to this contract.

Leonardo da Vinci: Patience serves as a protection against wrongs as clothes do against cold.
English saying: The devil is in the detail.
German saying: Der Teufel steckt im Detail.
Danish saying: Djævlen ligger i detaljen.
When mistakes are made, the mistake often lies in the detail.

Murphy's law. Anything that can go wrong will go wrong.
Opposite to Murphy's law: If something has gone well, then it is because you have overlooked something. (in danish Den omvendte Murphy: Hvis noget er gået godt, så er det fordi, du har overset noget).

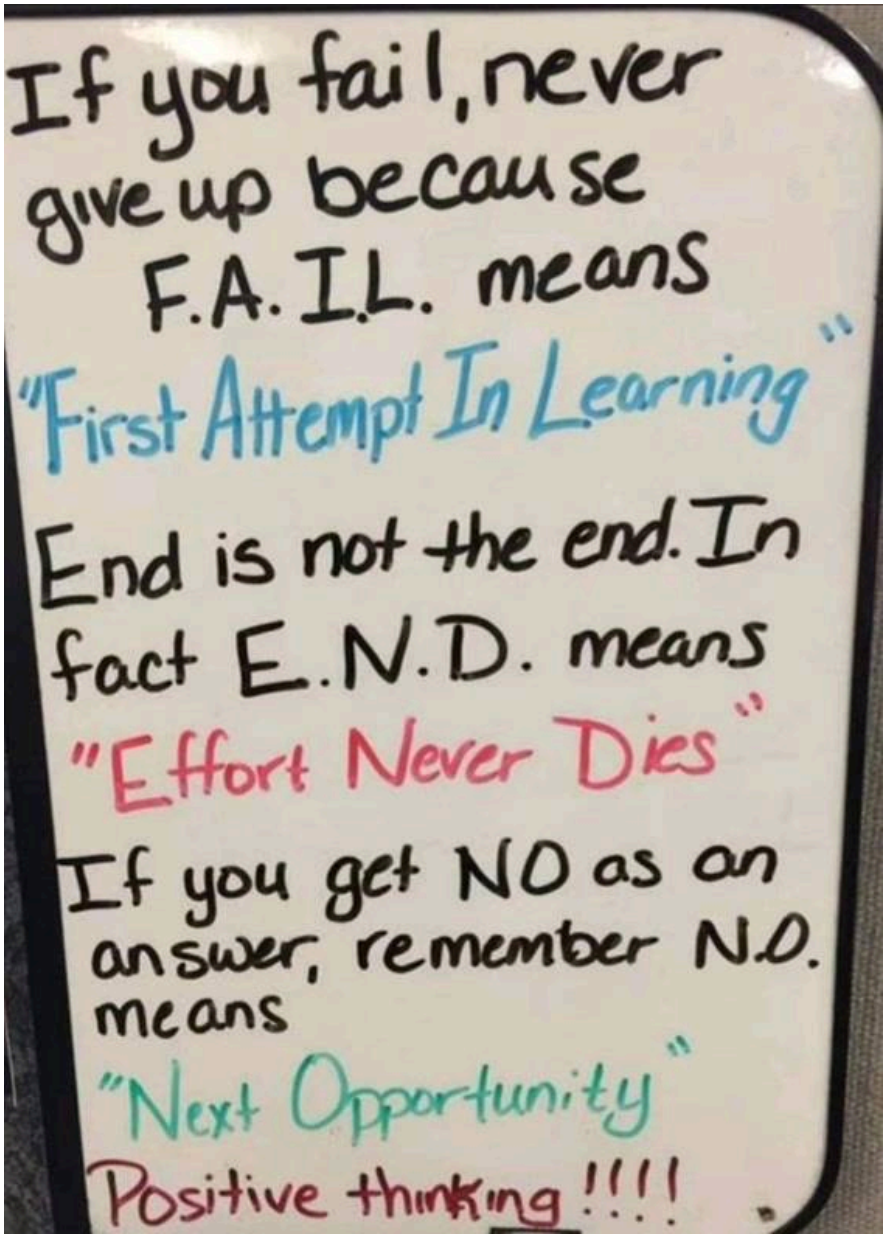
There are known knowns, means there are things we know we know.
There are known unknowns, means there are things we do not know.
There are unknown unknowns, means there are things we don’t know we don’t know. [United States Secretary of Defense Donald Rumsfeld, February 12, 2002]

How to find out that you know each other without knowing it?
Hvordan finder man ud af, at man kender hinanden uden at vide det?

To take a reservation of a car is not the same as to hold a reservation of a car.

Debtor, the one who owes money e.g. a customer for an item (in danish debitor, den der skylder penge).
Creditor, the one who has money to credit e.g. a seller who has sold an item (in danish kreditor, den der har penge tilgode).

Keep me posted, with a post notification (in danish hold mig underrettet med en underretning, let me know).



July 7, 2019, Pope Francis: »Our prayer should not be limited only to our needs and our necessities. A prayer is truly Christian if it also has a universal dimension.«