Trivadis White Paper

Comparison of Data Modeling Methods for a Core Data Warehouse

Dani Schnider
Adriano Martino
Maren Eschermann

June 2014

Table of Contents

Sources and Links

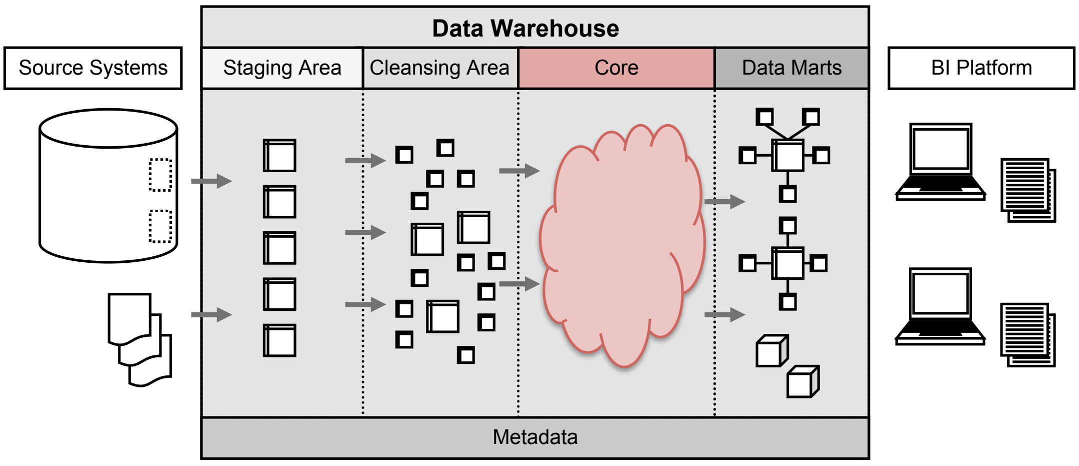
[1] Lawrence Corr: Agile Data Warehouse Design – Collaborative Dimensional Modeling from Whiteboard to Star Schema, DecisionOne Press, 2011, ISBN 978-0956817204
[2] Ralph Kimball, Margy Ross: The Data Warehouse Toolkit, Second Edition, Wiley & Sons, Inc., 2002, ISBN 978-0471200246, http://www.kimballgroup.com
[3] Symmetry Corporation: Getting Started with ADAPT, OLAP Database Design, 1998-2006, http://www.symcorp.com/downloads/ADAPT_white_paper.pdf
[4] C. Jordan, D. Schnider, J. Wehner, P. Welker: Data Warehousing mit Oracle – Business Intelligence in der Praxis, Hanser Verlag, 2011, ISBN 978-3-446-42562-0, p. 143 – 149
[5] Hans Hultgren: Modeling the Agile Data Warehouse with Data Vault, Brighton Hamilton, 2012, ISBN 978-0615723082, http://hanshultgren.wordpress.com
[6] Data Vault Challenge Workshop, Trivadis Blog, October 2013, http://blog.trivadis.com/b/danischnider/archive/2013/10/06/data-vault-challenge-workshop.aspx

This white paper describes some common data modeling methods used for the Core data model in Data Warehouses. The purpose of this document is to give an overview of the different approaches and to compare their benefits and issues. The comparison and recommendations at the end of the document should help to decide what method is appropriate for the requirements of a particular Data Warehouse project.

1. Introduction

In a Data Warehouse it is recommended to implement a central Core layer. Its purpose is the integration of data from multiple source systems and the historization of the data. The Core is the only source for the Data Marts that are used for BI applications such as reporting platforms or OLAP tools.

The Data Marts are usually designed as a dimensional data model with dimensions and facts. This model is either implemented as a star schema or as a multidimensional cube. But what kind of data model is appropriate for the Core layer? There is no best approach that suits for all Data Warehouses, but different modeling techniques are commonly used. Each method described in this paper has its advantages depending on the requirements and the modeling strategy.



The following chapters compare some of the common used Core data modeling methods:

- **Dimensional Core Modeling** with dimensions and fact tables
- **Relational Core Modeling** with master data versioning
- **Data Vault Modeling** with Hubs, Satellites and Links

Of course, additional methods or combinations of the different approaches are possible, too. We explain only the approaches that we use in customer projects and where we have practical experience.

The data models for other layers of a Data Warehouse (e.g. Staging Area, Cleansing Area and Data Marts) are not in scope of this white paper.

2. Aspects of Data Warehouse Core Modeling

The following aspects are important for each Core modeling approach. When comparing the different methods, we will focus on these topics and decide for each method how easy or complex it is to support these requirements.

2.1 Modeling Strategy

The Core data model can either be derived from the known analytical requirements (which sometimes are very specific requirements for reports) or from the data models of the source systems. These two strategies are known as Reporting-Driven and Data-Driven Analysis (see reference [1]).

In an ideal world the Core layer design is neither driven by specific source systems nor by currently known report requirements but by the definition of your business and your overall analytical requirements. With a pure data-driven strategy, the Core model must always be changed whenever the source system changes. With a reporting-driven strategy, you will not be able to fulfill new requirements without changing the Core data model.

The data model of the Core should ideally reflect the business processes in the best possible way. This is often a combination of the two strategies. For a stable and flexible Core data model, the current source systems as well as the known analytical requirements should be considered when designing the Core data model.

2.2 Data Integration and ETL

Data that is delivered from one or more source systems to the Data Warehouse must be integrated before it can be stored in the Core. Data integration is usually performed in ETL processes in the Staging and Cleansing Area to transform the data into a form that can be loaded into the Core data model.

The transformations for data cleansing and harmonization can be very complex, but are mainly independent from the Core modeling approach. They are more related to the OLTP data models and the data quality of the source data. Therefore, these tasks are not in scope of this document. However, the complexity of data integration from different source systems into a single Core model and the conversion steps between OLTP data models and Core target data model primarily depend on the modeling approach. This is one of the criteria to be considered when comparing the different modeling methods.

2.3 Historization

Most analytical requirements need historic data. Comparison of current data with data of the previous year, quarter or month are quite common to detect trends and necessary for planning and predictions. Additional requirements for storing historical data in the Core are traceability of data changes, repeatability of user queries, or point in time views of historical data.

The impact of the requirements above is that data in the Core must be stored in a way that supports change tracking and data versioning. Existing data must not be deleted or overwritten, but new rows have to be created for changes of existing source records. The behavior can be different for master data (dimensions) and transactional data (facts). Depending on the modeling approach, this can be implemented in different ways.

2.4 Lifecycle Management

New analytical requirements and changes in the source systems result in adapting the Data Warehouse accordingly. The robustness against changes in the analytical layer and/or source systems is a crucial requirement, especially when business requires agility.

For volatile layers (Staging Area, Cleansing Area) it is easy to drop and recreate the changed tables. Even for Data Marts it is possible to rebuild them from scratch – as long as the required data is available in the Core. But for the Core data model, the handling of change requests is a potential issue. New tables or attributes must be included in the existing data model, and the historical data has to be migrated or enhanced because of the changes.

All these situations as well as combinations of them occur in every Data Warehouse project on a regular base. Some change requests are just extensions of the data model and therefore easy to implement, other changes require a redesign and migration of the existing structures and are therefore more complex. The choice of the Core modeling approach can have an impact on how easy or complex it is to handle structure changes in a Data Warehouse.

2.5 Data Delivery to Data Marts

All persistent Data Marts of a Data Warehouse are loaded from the central Core. That implies that the Core must contain all relevant information for the dimensions and fact tables of the Data Marts, including historical data that is required by the users. If the Data Marts are implemented just as a view layer on the Core ("Virtual Data Marts"), this precondition is obvious. In an ideal Data Warehouse it is always possible to rebuild a Data Mart from scratch, or to create a complete new Data Mart easily.

The Core data model must be flexible enough to provide initial and incremental loads of dimension tables (SCD1 and SCD2) as well as fact tables of a dimensional Data Mart. This is possible if the Core is able to store a full history of all source data changes (see historization requirements above). Depending on the approach for historization, the loading steps for dimensions and facts can be more or less complex.

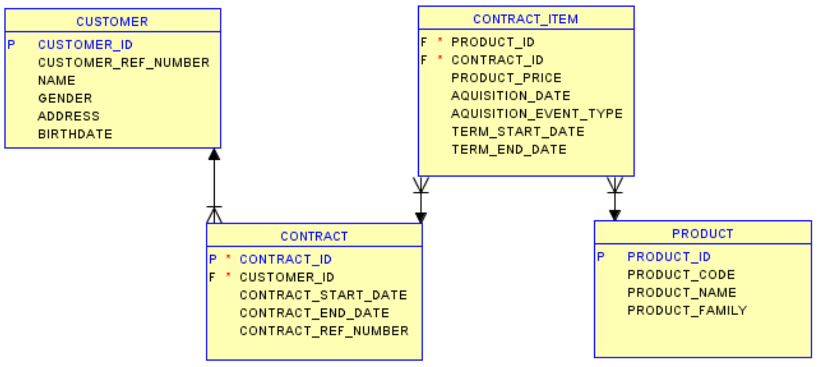
If a Data Warehouse architecture has a persistent Data Mart layer, load processes are required to load data from the Core layer to a Data Mart. The complexity of these load processes depend on the modeling approach of your core layer, but also on some other design decisions.

3. Case Study

To compare the different modeling approaches in this white paper, we use a fictitious case study adapted from a real example of an insurance company. The Data Warehouse contains data from two source systems and delivers information to a dimensional Data Mart based on known business requirements. During the project lifecycle, one of the source systems is changed in a way that affects the Core data model of the Data Warehouse. The OLTP data models of the source systems and the business requirements are described in the following sections.

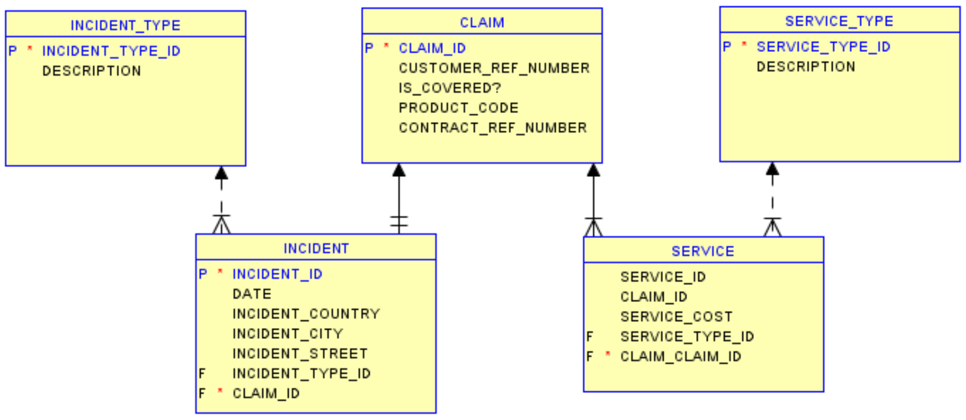
3.1 CRM Source System

One of the source systems for the Data Warehouse is a Customer Relationship Management (CRM) system. It contains information about the customers and their contracts with the insurance company. Each customer can have many contracts, but each contract is defined for exactly one customer. A contract can contain one or more products. For each product, a contract item is created in the CRM system. There are three possible events in the CRM system for a contract item: acquisition, renewal and termination of a contract item. The data model of the CRM system contains four tables and is displayed in the following entity-relationship diagram:



3.2 Claims Source System

The second source system is the Claims system that manages incidents and claims of the insurance company. A claim is the declaration to the insurance. An incident is the reason for a claim. One claim can have only one incident. Services are provided under the coverage of the insurance (lawyer, towing of a car, car repair, …). Each claim can generate one or many service(s). The entity-relationship diagram of the Claims system has the following structure:



3.3 Business Requirements

Several user groups will work with Business Intelligence applications on a Data Mart. Each group of people has its own requirements:

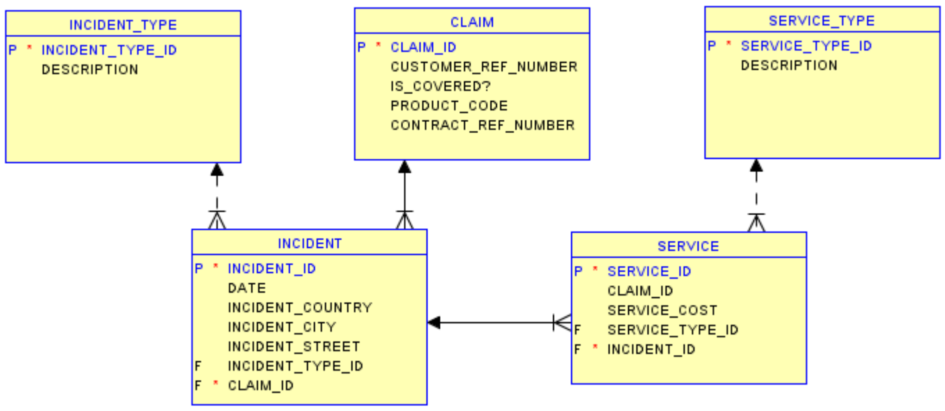
- **Employees controlling claims** want to have reports giving statistics about the claims and incidents. The need to analyze cost of coverage and number of claims
  - By place of incident
  - By type of incident
  - By time
  - By product

- **Managers** want to have a global view. They need to analyze the cost of coverage and the generated income
  - By client (country, city ,age, gender)
  - By product
  - By time

- **Marketing personal** need to analyze the number of acquisitions, renewals and contract terminations as well as the number of contracts
  - By client (country, city ,age, gender)
  - By product
  - By time

3.4 Change Scenario

Source system changes are typical when maintaining a Data Warehouse, which cannot be robust against all kind of changes. In our case study, the Claims system is subject to change . since the underlying business process has been changed. The cardinality of the relation between claim and incident has been adapted, i.e. one claim can contain multiple incidents. An impact of this process change is that services are now related to incidents, not to claims anymore. For an incident, one or more services can be attached.



4. Dimensional Core Modeling

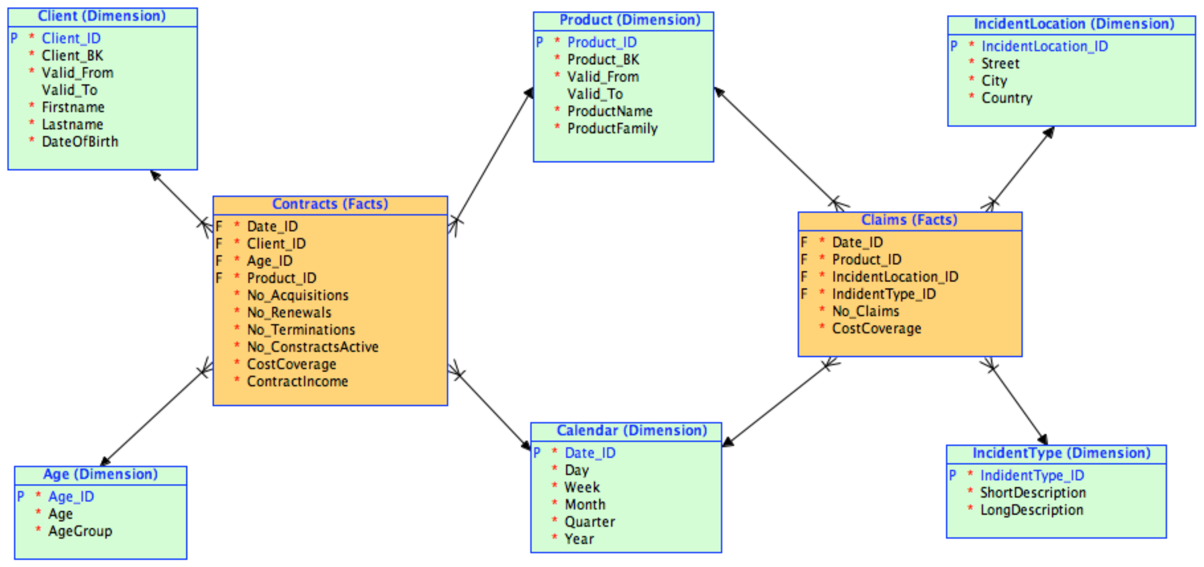
Dimensional modeling is a set of techniques and concepts used in Data Warehouse design proposed by Ralph Kimball (see [2]) which is oriented around *understandability* and *performance* and which uses the concepts of facts (measures), and dimensions (context). Most reporting tools and frameworks require a dimensional model, business users intuitively understand a dimensional model and are able to formulate queries.

A dimensional model is a conceptual model rather than a logical/physical data model, since it can be used for any physical form, e.g. as relational or multidimensional database. If a dimensional model is implemented in a relational database, the most commonly used schema types are star and snowflake schema. The latter one normalizes dimension tables, in a star schema the dimension tables are denormalized. The transition from a dimensional to a logical/physical data model is straight forward and should follow well defined guidelines.

To make the different model approaches of this white paper comparable, the dimensional model is presented as star schema. Nevertheless, it is highly recommended to use a dimensional model for discussions with business users, documented by using a common notation like the ADAPT Notation (see [3]).

4.1 Modeling Strategy

A dimensional model is driven by known and well defined analytical needs. The following star schema (in the following referenced as star schema 1) fulfills all report requirements of the case study:



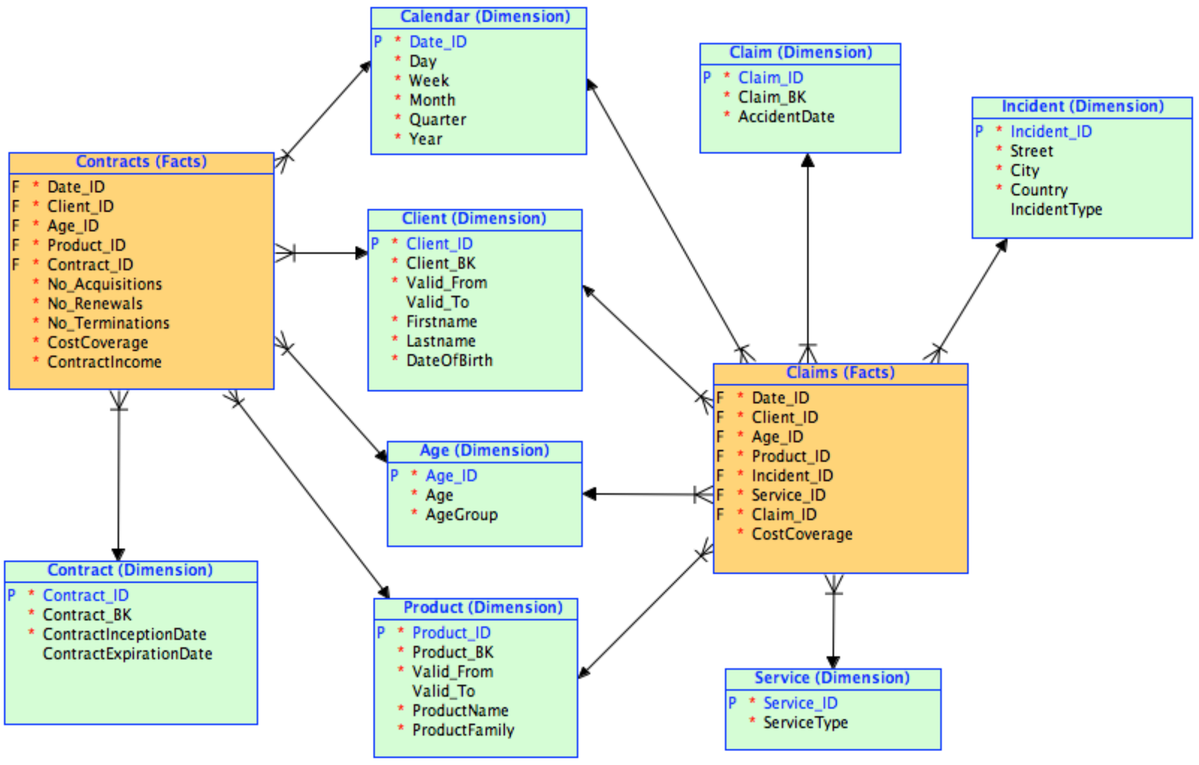
The star schema has *less information* than the original source data model. It has no information about a single claim or contract, nor does it contain any information about services, since the required reports do not need this information.

When designing a star schema it is a valid approach to simplify, reduce information and pre-calculate certain measures, e.g. counting business entities, to derive a model, which is easy to understand, which answers the given analytical questions and which is optimized regarding performance. The latter is especially important when your Data Mart layer is only a “thin” view layer or if your warehouse architecture has no Data Mart layer at all.

The drawback is that new, unconsidered analytical questions cannot be answered:

- The model does not contain any information about the service type. Analyzing costs per service type would require sourcing additional tables and a modification of the model.
- The model is not able to answer questions like “what is the average number of products in a contract”, since the star schema does not contain information about single contracts. The same is true for claims.
- The model is not able to answer questions “what is the average price of a product”, since the star schema does not contain information about the price for a single product.

Since interesting analytical discoveries often result in even more and new analytical questions, the above star schema – although fulfilling the given analytical needs – might turn out to be too restrictive in the long run. Therefore, the source system should be considered as well, if it is conceivable that additional requirements will have to be implemented.



This model (in the following referenced as star schema 2) is able to answer a lot more analytical questions, but still has limitations and drawbacks compared with star schema 1.

- Higher degree of complexity: additional dimension tables, more complex fact tables
- Larger fact tables due to higher granularity
- Querying the number of contracts/claims requires a “Count distinct” operation
- The relation between contract and product and between claim, service and incident is implicitly contained in the fact table. Questions like “how often does a price change during the last three years” can be answered, but the model is far from optimal for this kind of analytics due to the query complexity.

Designing a dimensional model always is a very difficult tradeoff between simplicity and analytical flexibility. The more complex the source systems are, …

- … the more tables and relations have to be considered and the more fact tables and combinations of dimensions are possible,
- … the more information is lost due the transition from an entity-relation model to a dimensional model because modeling options of a dimensional model are restricted,
- … the more analytical questions may arise and the more difficult it will be to anticipate them.

Consequently the gap between the analytical capabilities of your model and the future analytical needs will become larger with increasing complexity of your business and your source systems.

4.2 Data Integration and ETL

The complexity of the ETL processes into a (dimensional) Core model is determined by several factors, such as the number of source systems and their complexity as well as the complexity of the integration and business transformation steps. Loading dimension tables usually is less complex than loading the fact tables, since less source tables have to be considered.

- Loading the product dimension table is easy since only one single source table is used.
- Loading the client dimension table is more complex since data two source systems have to be integrated.
- Fact table 1 of star schema 1: ETL requires
  - Reading all contract items and count all those, which have been acquired, renewed or terminated during the period to be loaded, the linkage to client and product is straight forward, but the age has to be calculated.
  - Reading all contracts and count all active contracts
  - Reading all covered claims where the corresponding incident occurred during the period to be loaded and aggregate the costs of the corresponding services

Very often the ETL complexity for fact tables is rather high and usually requires transformation and integration of numerous source tables. Since a star schema is completely different than the typical OLTP data model, data model restructuring is highly complex and often requires multiple steps and intermediate tables to cope with that complexity and to achieve the necessary load performance.

4.3 Historization

Kimball introduced the concept of slowly changing dimensions (SCD) in 1996. To deal with the various historization requirements different SCD types may be implemented. The most common types are SCD1 (overwrite old with new data) and SCD2 (a new record is created for each change). SCD2 dimension tables have two additional timestamps indicating the validity of a record. In our example, the client and product dimensions contain SCD2 attributes and therefore contain additional attributes for the validity range.

The historization concept in a dimensional model is strongly interrelated with its key concept. Whereas the two other model approaches assign a surrogate key to each entity, in a star/snowflake schema a surrogate key is assigned to each new *version* of an entity. Therefore, since the surrogate key also contains information about validity information, joining fact and dimension tables can be done without considering any timestamp attributes, resulting in join operations which are more intuitive for users and can be more efficiently handled by the database.

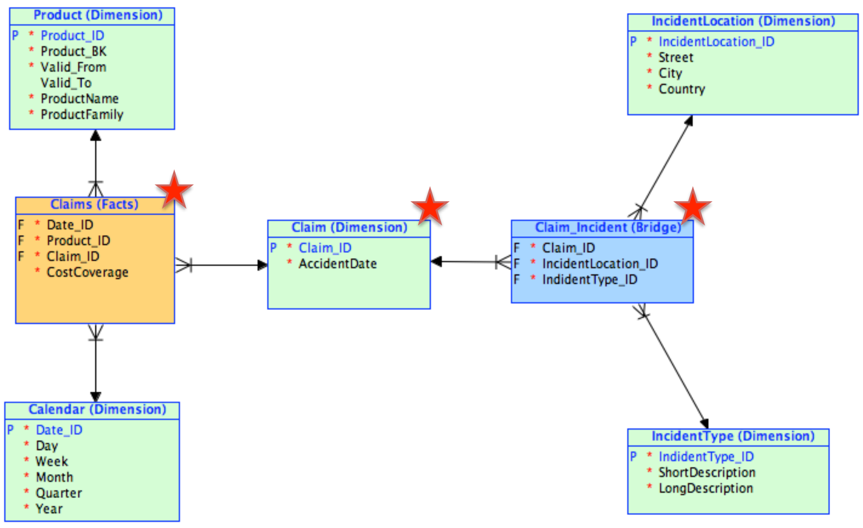
Having different primary keys for each version has one drawback: the so called *ripple effect*. Each new version of a parent "triggers" new versions in all children, which might lead to an "explosion" of child elements. Example: If a product sub-category is assigned to a new product category, it gets a new primary key. Therefore, all products in this sub-category have to reference this new primary key, which in turn results in creating new version records with a new primary key for each affected product. In our example, the client and product dimensions contain SCD2 attributes and therefore contain additional attributes for the validity range.

4.4 Lifecycle Management

Star schema 1 and 2 both are based on the fact that there is a 1:1 relation between incident and claim, however this is not enforced by any of the two models (e.g. by database constraints). Both schemas can remain unchanged if it is possible to either allocate the overall claim costs to multiple incidents or to define one single incident as "primary incident". The latter solution results in losing the information about the other incidents of a claim. Of course, both solutions would require adapting the ETL processes, implementing the allocation process or the determination of the "primary incident" based on specified business logic. None of the solutions would require any data migration because the model remains unchanged.

If none of the above approaches is possible, a 1:m relation between fact table and dimension has to be realized, which is not straight forward in a dimensional model. One possible modeling option is a bridge table, which results in a more complex model and ETL processes.

The revised star schema 1 looks quite different, all affected/new tables are marked in the model below. The revised fact table now contains much more records (one record for each claim) due to a higher granularity. Data migration is not possible, since the original star schema did not contain any claim data.



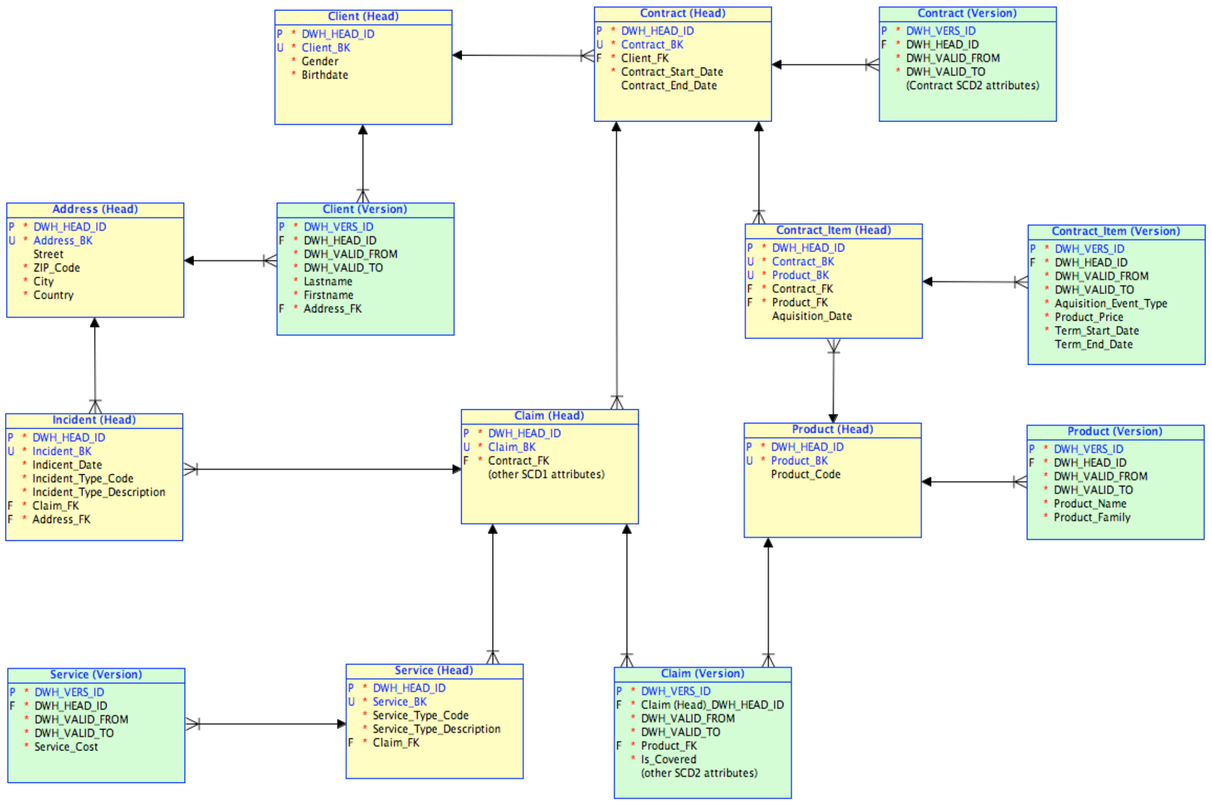
4.5 Data Delivery to Data Marts

Data Warehouse architectures with a dimensional core do not necessarily require a persistent Data Mart layer. A separate Data Mart layer may be required if a multidimensional database is in place or if Data Marts in a lower granularity compared to the data in the core layer are required. In the latter case the Data Mart layer may consist only of views, which can be materialized if the performance is not satisfactory. Complex ETL processes from core to Data Mart are not necessary since both layers follow the same modeling paradigm.

5. Relational Core Modeling

Relational data modeling is usually used for operational systems (OLTP systems) that store their data on relational databases. But it can only be used for Data Warehouses – most structured DWH databases are implemented with relational database systems. The term „relational“ to describe the modeling approach of a Core data model is a bit misleading because all of the methods described in this paper use relational databases. Even the term „3NF model“ (for a data model in 3rd normal form) does not clearly define this approach. When a dimensional model is implemented as a snowflake schema, it is in 3rd normal form, too.

The following data model is a suggestion of a relational Core model for our case study. The different aspects about modeling strategy, data integration, historization lifecycle management and data delivery to Data Marts are described in the next sections.



A relational Core data model is not necessarily a 3NF data model. In our case study, the code tables for incident type and service type are denormalized in the Core.

5.1 Modeling Strategy

A relational Core data model is often used for Data Warehouses when the user requirements for reporting or other BI applications are not yet defined or when the same information is required for different purposes. For example, the age of a client in our example is currently used as a dimension in the dimensional model of the previous chapter. But probably it will be used as a fact in another Data Mart, for example to derive the average age of clients per claim. A relational Core model allows more flexibility in the usage of the same data in different Data Marts.

The modeling strategy for a relational Core model is often data-driven or a combination of data-driven and reporting-driven. The Core data model is derived from the source systems, but known business requirements should be considered where possible.

Our example Core model is derived from the data models of the two source systems. The known business requirements are considered in the Core model, i.e. it is easily possible to deliver the required information to a dimensional Data Mart with the facts and dimensions described in the previous chapter.

5.2 Data Integration and ETL

An important characteristic of a Core data model is that data from different sources is integrated. The model has to be subject-oriented, not a one-to-one copy of the source tables. Similar data is stored in one place of the Core. An example for our case study is the Address table. It contains customer addresses of the CRM system, but also the places of incident of the Claims system.

The complexity of the ETL processes depends on the transformation rules between source system and Core data model. If the Core model is very similar to the data model of the source system – this is usually the case if only one (main) source system is used -, the transformations are typically easy. If multiple source systems have to be integrated in one common Core data model, the complexity will increase.

The ETL processes for Master Data Versioning (see next section) are not trivial, but easy to implement because the logic is always the same, independent of the data contents. They can easily be generated or implemented with operations of an adequate ETL tool. An established method is to implement the (complex) data integration transformations between Staging Area and Cleansing Area. In this case, the Cleansing Area has the same structure as the Core data model, but without historization. Then, the ETL processes between Cleansing Area and Core are only used for Master Data Versioning and assignment of surrogate keys.

5.3 Historization

For flexibility to load SCD1 and SCD2 dimensions in the Data Marts, master data (i.e. the descriptive data used in dimensions) must be versioned in the Core data model. Data versioning could be realized with SCD2 like in the dimensional model, but for relationships between entities, this can lead to many different versions when existing data of a referred entity is changed (see „ripple effect“ described in chapter 4.3). With the concept of **Master Data Versioning**, this ripple effect can be avoided. For each entity in the relational data model, two tables are defined:

- The **head table** contains the identification (business key or primary key of the source system) and all attributes that either cannot change during their lifecycle, or where only the current state is relevant (SCD1). These can also be relationships to other entities, e.g. the relationship from a contract to a client.
- The **version table** contains all attributes and relationships that can change during their lifecycle, and where the change history is required in at least one Data Mart (SCD2). Each version has a validity range (valid from / valid to) and refers to the entity in the corresponding head table.

In our example, the relationship between a contract and a client cannot change and is therefore stored in the head table of the contract. The relationship between client and address can change during the lifecycle of the client. Thus, the foreign key column to address is stored in the version table of the client.

A version table is only defined when at least one attribute or relationship requires change history tracking. This is not the case for addresses (we are interested only in current addresses) and for incidents (an incident happens once and is not changed afterwards).

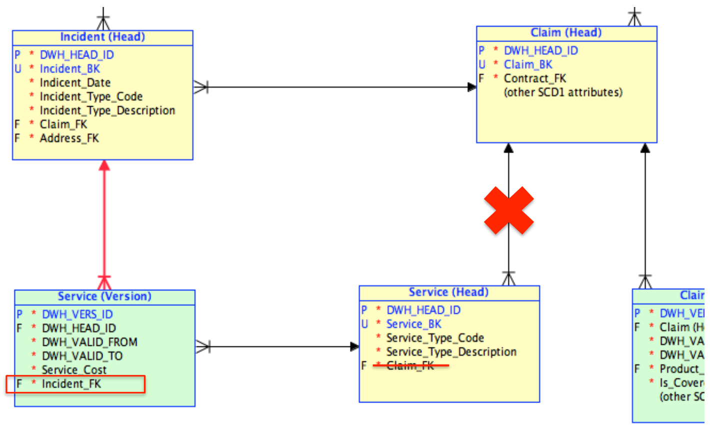
The separation of head and version tables is required to store independent change history for each entity. Although a foreign key can be static (stored in the head table) or dynamic (stored in the version table), it always refers to the head table of the corresponding entity, not to a particular version. This allows a high flexibility for data historization in the Core and prevents the ripple effect described above.

An important difference between Master Data Versioning and Slowly Changing Dimensions Type 2 is the behavior of key references: A reference to an SCD2 dimension is always related to the particular version that is valid at the event time (or load time) of a fact. Relationships between entities in a relational Core model refer to the entity key (i.e. the primary key of the head table), not to a particular version. The version key is implemented for technical reasons only, but never referred. The decision what version is relevant is done at query time or when a Data Mart is loaded. See chapter 5.5 for details.

5.4 Lifecycle Management

When the relational Core mode is derived from the OLTP data models, structure changes of a source system require modifications of the Core model. In our example, the Claims source system is changed, so that a claim can contain multiple incidents, and a service is attached to an incident, not directly to the claim anymore.

Because the existing relationship between incidents and claims in the Core already allow to store multiple incidents per claim, no model changes are required for this part. But the relationship from services to claim must be changed to a relationship to the incident head table. Let's assume that a service can be transferred from one incident to another within the same claim. In this case, the foreign key column to the referred incident must be stored in the version table of the service.



Adding new entities or attributes to a relational Core data model is usually not very complex. But when existing structures – especially relationships – are changed, the effort for data migration in the Core may be high. For our example, it is simple to add a new foreign key column to the service version table. But what contents must be filled in for the historical data records? For each service, we can identify the corresponding claim. If only one incident exists for this claim (and this was guaranteed in the initial data model of the source system), the foreign key to the correct incident can be determined clearly. All current and previous versions of all services must updated with the correct incident before the foreign key column and relationship between service and claim is eliminated

Unfortunately, not all model changes allow a complete and accurate data migration. If the previous Claim system already allowed to have multiple incidents for one claim, a definite assignment of an incident for each service would not be possible. If a new attribute is added to the system, no information about its contents in available for historical data. In such situations, default values or singletons must be assigned, and the business users must be aware that the required information is only available for claims after the release data of the changed source system.

5.5 Data Delivery to Data Marts

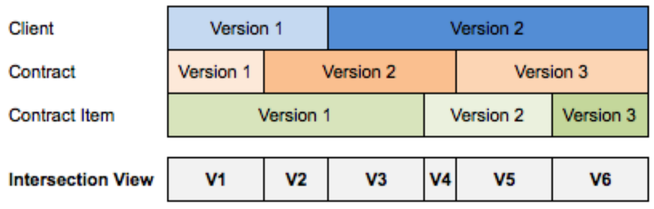
To load the (dimensional) Data Marts from a relational Core model, the data must be transformed into dimensions and facts. For example, we have to load a Claim dimension with information about claims, incidents, services and products. Before an appropriate ETL process to load this dimension table can be implemented, we have to define the granularity of the dimension, i.e. the detail level of the data. In the Core data model, the most detailed level are the services. If the granularity of the Data Mart dimension is a claim, incidents and services can only be stored in an aggregated form. For example, one „main service“ must be identified, or the number of services is stored in the dimension table. Another option is to skip the information about services in the Claim dimension and to identify a separate Service dimension. The relationship between claims and services is then only possible through the fact table.

To avoid complex ETL processes to load the Data Marts, its is recommended to „hide“ the Master Data Versioning with a History View Layer on the Core tables. For each pair of head and version table, the following views are created (or better generated):

- The **Current View** contains all business attributes of head and version table, but shows only the current state of the data. This is implemented with an join of head and version table and a filter on the latest version. Optionally, a context driven view can be implemented that allows to define a „query date“ with a parameter or context variable. Current views are used to load SCD1 dimensions or for incremental loads of SCD2 dimensions.

- The **Version View** contains all business attributes of head and version table and the validity range (valid from / valid to) for all existing versions. Version views are used for initial loads of SCD2 dimensions. When only one version view is involved to load a dimension table, this is straightforward. But when two or more version views must be combined to load one SCD2 dimension, an additional intersection view is required.

- The **Intersection View** determines all intermediate versions based on the combinations of the validity ranges of all version views. A possible implementation is described in [4]. This view is used for initial loads of SCD2 dimensions with data from multiple head and version tables.



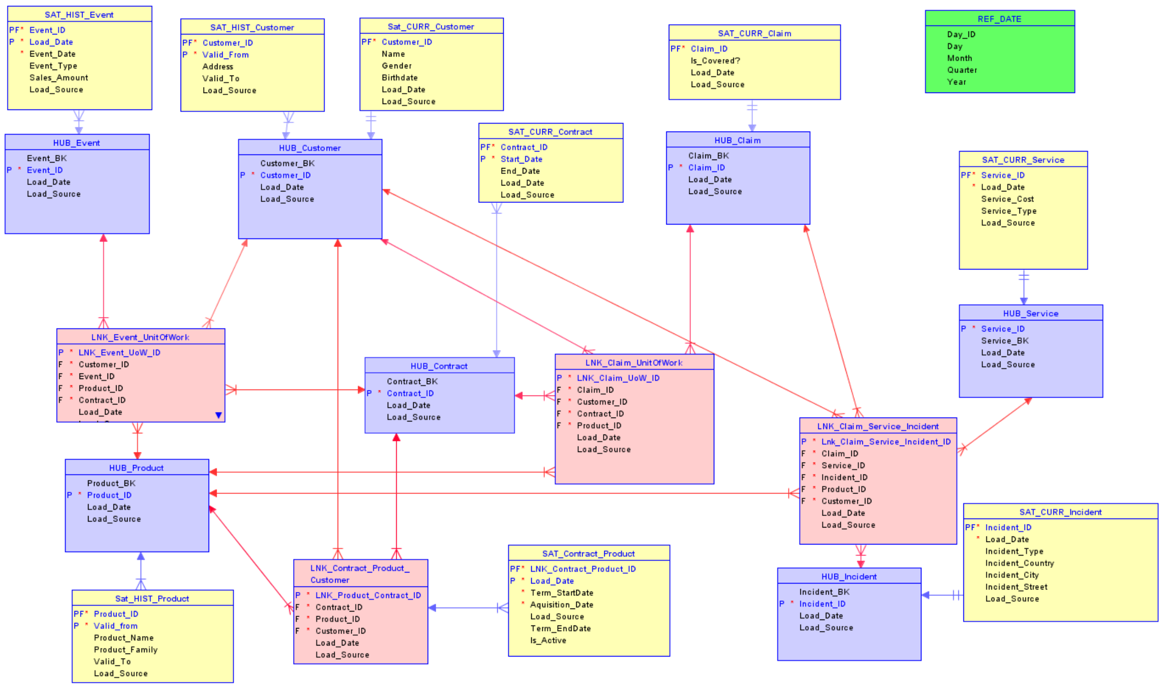
With a combination of Master Data and a History View Layer, the complexity to load a dimensional Data Mart can be reduced. But if the relational Core data model is very different from the required dimensions and facts in a Data Mart, the additional transformations required to load the Data Mart can be rather complex.

6. Data Vault Modeling

The Data Vault modeling, invented by Dan Linstedt in 1990, is used to model the enterprise Data Warehouse Core layer. This approach is suitable for multi-source environments needing a fast adaptation to changes. The model consists of a complete denormalization ("*Unified Decomposition*" mentioned by Hans Hultgren, see [5]) of three base constructs into entities:

- A **Hub** is the business key defining a core business concept. This key is related to a business entity, not to a particular source system.
- A **Link** is the business natural relationship between business core entities.
- A **Satellite** is the contextual information around keys and relations. It contains the descriptive information of a business entity.

This split allows your Enterprise Data Warehouse to have a quicker adaptation to business need, source changes or business rules, though with the disadvantage of an increasing number of structures in the core Data Warehouse.



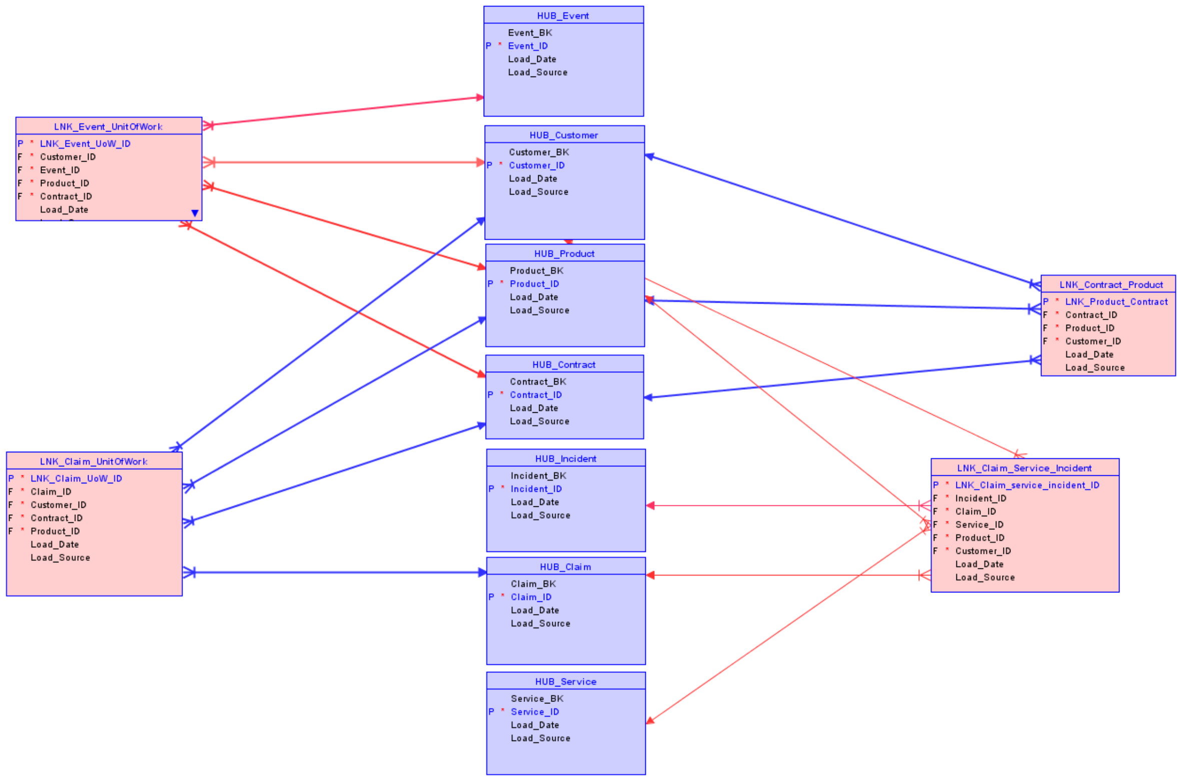
6.1 Modeling Strategy

In Data Vault, discussions with the business are mandatory to identify the business core concepts and the processes used for Hubs and Links. Interpretations of the source systems are possible, but sometimes source systems are not providing the truth about business reality. The core of the model has to be independent from the sources to guarantee the longevity of the enterprise core Data Warehouse and minimize dependencies with the source systems.

The **Hub** structure contains the business core concepts of a functional area. In the design above, we can find seven Hubs, each Hub contains a business key and an internal DWH surrogate key. The business key is the enterprise wide identifier, the key used by the business to isolate a singleton of each business concept. The surrogate key is used with the Data Vault model for references. Transactions and events have to be modeled as independent Hubs; they are core concept of the business. In our example, the customer business key would probably be the CUSTOMER_REF_NUMBER, as long as this identifier is understandable by the business and known as the unique identifier.

Links are the Hub surrogate key associations. There is no limitation for the number of referenced Hubs as long as the association is a business relationship. The physical entity contains the different surrogate keys of the linked Hubs, each in a separate column, and a surrogate key for the association. The Link concept offers a lot of possibilities, the structure is independent and can help to define new relations or meet new needs. In our example, LNK_Event_UnitOfWork represents the natural association for an event. An event will always have a customer, a contract and a product. For Claim, it has to be verified with the business that a claim will always have a customer, a contract and a product. Same for Service: A service will always have an incident, a claim, a product and a customer. This has to be verified with the business.

If information at the incident level for claim is needed, a new Link has to be created between those two entities, otherwise a “select distinct” on the Link will be necessary to get the correct granularity, and this is not acceptable. Again, the Data Vault model allows you to create Links without impacting the rest of your model. Here is a simplified view (without Satellites) of the Data Vault model:



The **Satellites** contain all the information describing the business keys or the associations. There can be many Satellites for one Hub or one Link, the idea is to regroup attributes by frequency of changes or/and by source. Each Satellite entity contains a surrogate key linking to a business key in the Hub or to a Link. The particularity of the Satellite is that the load date is part of Satellite key.

Each time an attribute change is loaded, a new row is inserted with the load date of the Satellites insertion. In our example, there are two Satellites for the Hub customer because we regroup attributes by frequency of changes. In the Satellite SAT_CURR_Customer, we have name, gender and birthdate that will most likely not change and where the business only need the current version of the record. The Satellite SAT_HIST_Customer contains the attributes whose changes have to be tracked by the DWH, for example the customer’s place of residence.

6.2 Data Integration and ETL

Data integration of multiple source systems into one DWH is always hard work. For Data Vault, the strategy will be to find the core business concepts and the natural business keys in the source systems and then verify that they confirm to the business point of view. The same approach is used for the relations.

The flexibility of the Data Vault model is due to the fact that it allows adding objects without a complete redesign of the existing structure and the possibility to have different Satellites for the source systems . This strategy allows tracking of the differences and quality job into the core DWH **but** postpone the sourcing choice of the attributes to the Data Mart layer. For example, if a new source system , also containing customer entities, has to be loaded into the Data Warehouse, a new Satellite with different attributes can be added to the model. A simple report comparing the two Satellites will show the quality issues to the business.

For quality problems during the integration, a quality Satellite can be added, containing the details of the integration issues. For example, there is a customer that is in the claim system but not in the CRM. As the core concept is the same, a step in the ETL is added to load the claim customer key into the Hub. The first load source of a key will be the claim system, and an entry will appear in SAT_Customer_Quality that this key is 'Not in the CRM system'.

The ETL jobs to load a Data Vault typically run in two steps: In a first step, all Hubs are loaded in parallel. In a second step, all Links and Satellites are loaded in parallel. The individual ETL operations for each type of objects are:

- Hubs: The business key must appear only once in the Hub; insert with a lookup on the business key
- Links: The association must appear only one time in the Link; insert with a lookup on the association of surrogate key
- Satellites: The loading of the object is the most complex, it depends on the historization mode. Either the changed attributes are just updated, or a new version must be inserted.

Every object (Hub, Link, Satellite) in the Data Vault methodology stores two auditing attributes: The first load date (i.e. the first date when the specific line appeared) and the first load source (i.e. the first source where the specific line appeared). They are providing audit information at a very detailed level.

6.3 Historization

In the Data Vault model, historization is located in the Satellites. It looks like SCD1 or SCD2 dimensions, except that the primary key of the Satellite is the surrogate key of your Hub and the load date of the new record in the Satellite. This historization approach is used including the facts (facts are Hubs and have Satellites).

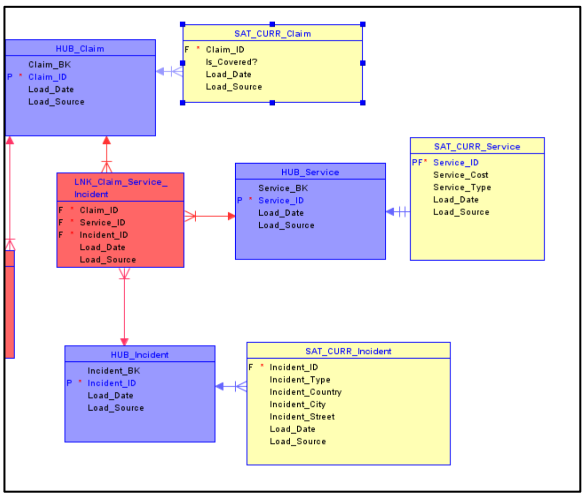
As already explained, it is possible to have multiple Satellites per Hub/per Link grouping them by frequency of change. The multiple Satellites strategy can become complicated when multiple Satellites are historized, the complexity is to regenerate time slices within multiple historizations. In this case, there is a possible solution called "point-in-time" table, the structure placed between the Hub/Links and the historized Satellites stores the time slice reconciliation and the keys. This reconciliation is heavy: every time there is a change in one of the Satellites, a new record in the point-in-time table linking each Satellite (surrogate key of each Satellites + load date of each Satellites) with the current time slice record is created.

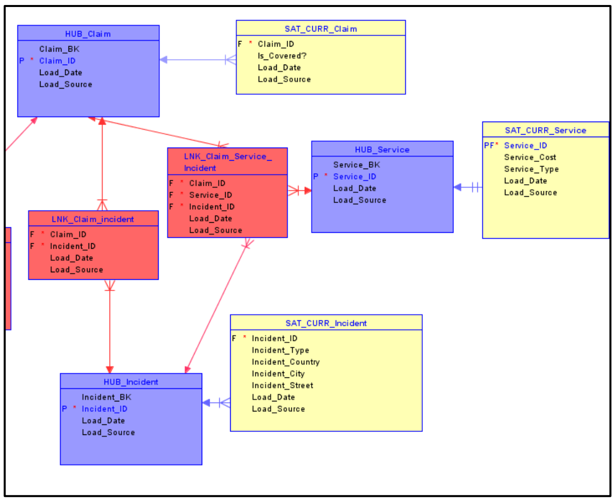
6.4 Lifecycle Management

The "Unified Decomposition" offers the possibility to adapt/change/create tables without touching to the core structure of your Data Warehouse. Based on the insurance case, it is possible to describe some practical lifecycle management scenarios:

- New attributes for the customer:
  - Add a Satellite with the new attributes: this method is very agile, there is no impact on the rest of the Data Vault model
  - Add the attributes in an existing Satellite: this method is heavier but, still, there is an impact only on the attributes in the same Satellite, the change is not impacting the Hubs or the Links.
- New Hub/Link/Satellite:
  - This change is transparent for the Data Vault, there is no impact on the core.

Our change scenario (see chapter 3.4) will have no impact on the Data Vault structure for the service analysis, this is the many to many relationship benefits. In business language, a service will always have an incident and a claim. The only change is that we have to add a new Link between Claim and Incident, this change does not impact other objects of the model.





6.5 Data Delivery to Data Marts

Generally, the Data Vault model fits in a Data Mart star model this way:

- Links become foreign keys in the fact tables
- The event Hubs are providing the granularity of the fact table while the other Hubs are becoming dimensions
- The Satellites are becoming facts or dimension attributes depending on the attached Hub (event or not)

Data delivery from a Core Data Vault model can generate performance issues, because of the multiple "joins" that are sometimes outer in case all the records are not in the Satellite of a Hub. If the model contains point-in-time tables, the queries to reconsolidate the historized dimensions or facts through multiple SCD2 Satellites is very complex and, again, increase the number of "joins" in the queries.

As seen in the previous chapters, all the information is physically stored in the Core Data Warehouse, every need can be generated from it.

7. Conclusion

7.1 Comparison Matrix

Topic	Dimensional	Relational	Data Vault
Complexity			
Overall architecture complexity	Simple and easy to understand	Separation of static and dynamic attributes and relationships	Requires in-depth understanding of Hubs, Links and Satellites
Intuitive and understandable data model	Low number of tables, data model near end user layer (Data Marts)	High number of tables due to head and version tables	Very high number of tables due to complete denormalization
Data Integration and ETL			
Integration of multiple source systems	Transformation rules for integration must be implemented in ETL processes	Transformation rules for integration must be implemented in ETL processes	Separate Satellite for each source system with common business key in Hub reduces complexity
Complexity of ETL processes to load Core	Transformations between OLTP models and dimensional Core can be complex	Transformation rules relatively simple when data model is similar to source system	Simple standard ETL rules to load Hubs, Links and Satellites
Historization			
Handling of data versioning (SCD2) of attributes	Easy delta detection between source system and dimension table	Easy delta detection between source system and version table	Easy delta detection between source system and Satellites, PIT tables for multiple Satellites
Handling of data versioning (SCD2) of relationships	"Ripple effect" for changes on higher hierarchy level	Same logic for relationships as for attributes	Quite simple located in the Satellite of the Link, for SCD support of Data Marts, PIT
Lifecycle Management			
Robustness against source data model changes	Source systems changes often have impact to Core data model	Source system changes require table changes in Core	Existing tables are not affected, only new Satellites required
Robustness against new analytical requirements	New requirements typically have impact to Core data model	Model changes required only if required data is not yet in available Core	No changes in Data Vault model, only data delivery to Data Marts must be adapted
Easy to change model	Refactoring of existing tables required in many situations	Historical data must be migrated in some situations	Existing tables are not affected, only new Satellites required
Data Delivery to Data Marts			
Initial / incremental load of SCD1 dimension	Very easy because of similar structure in Core and Data Mart	Easy by joining current views of History View Layer	Easy by reading current version of each Satellite
Initial / incremental load of SCD2 dimension	Very easy because of similar structure in Core and Data Mart	For initial loads, intersection views are required	Complex, if PIT tables are required for Hubs with multiple Satellites
Initial / incremental load of fact table	Very easy because of similar structure in Core and Data Mart	Easy because facts are related to event date and have no version table	Incremental load of large fact tables can be a performance challenge (see [6])

7.2 Recommendations

Each Data Warehouse is different. Not only the known (or unknown) analytical requirements and the number, complexity and structure of the source systems have an impact on the architecture and design of a Data Warehouse. Other aspects such as industry sector, company culture, used technologies and tools, data volumes and available knowledge in IT and business departments can have an influence on the recommended design methods.

As the comparison matrix on the previous page shows, none of the data modeling methods described in this white paper fits for everything. Each method has its benefits and issues. The following recommendations should give some guidelines what method is appropriate in which situation.

- **Dimensional modeling** is strongly recommended for Data Marts. But a dimensional model is also very appropriate for the Core data model of a Data Warehouse, if the analytical needs are known and well defined. A great advantage of a dimensional model is its understandability for business users. Data integration and ETL processes to load a dimensional Core are usually more complex compared to the other modeling methods, but contrariwise the processes to load the Data Marts are much easier than those of the other approaches. A recommended architecture pattern with a dimensional Core is that the Data Mart layer is not realized as a physical layer, but only as a set of views on the Core model. The project team must be aware of the fact, that new analytical needs usually require the adaptation of the Core data model – with all consequences.

- **Relational modeling** is useful for the Core layer if the analytical requirements are diffuse or not yet defined or if the purpose of the Data Warehouse is the delivery of information to different kind of BI applications and other target systems. In combination with Master Data Versioning, a relational Core gives flexible support of data historization and allows delivering current and historical data to the Data Marts. The ETL processes to load a relational Core are usually easier than for a dimensional Core, but the data delivery to Data Marts can be more complex – especially if the Core model is derived from the data models of the source systems. A relational Core is recommended if the source system structures are relatively stable and if a highly integrated Core data model is required.

- **Data Vault modeling** is a powerful approach for Data Warehouses with many source systems and regular structure changes on these systems. Adding new data to a Data Vault model is quite simple and fast, but the price for this is a complex data model which is harder to understand than a dimensional or relational data model. Instead of a full integration, data from different sources is assigned to common business keys, but stored in its original form. This allows simpler ETL processes to load data into the Core, but postpones the integration effort to downstream processes. Hence data delivery to the dimensional Data Marts is highly complex and costly. Data Vault modeling is a recommended method especially for agile project environments with short incremental release cycles.

Fundamental for all modeling approaches is that the method is accepted by the DWH development team, and all architects and developers understand the concepts, benefits and issues of the chosen modeling approach.