



Data Vault and Data Virtualization: Double Agility

A Technical Whitepaper

Rick F. van der Lans
Independent Business Intelligence Analyst
R20/Consultancy

March 2015

Sponsored by



CENTENIUM

Copyright © 2015 R20/Consultancy. All rights reserved. Cisco and the Cisco logo are trademarks or registered trademarks of Cisco and/or its affiliates in the U.S. or there countries. To view a list of Cisco trademarks, go to this URL: www.cisco.com/go/trademarks. Trademarks of companies referenced in this document are the sole property of their respective owners.

Table of Contents

1	Introduction	1
2	Data Vault in a Nutshell	2
3	The SuperNova Data Modeling Technique	8
4	Rules For the Data Vault Model	10
5	The Sample Data Vault	12
6	Developing the SuperNova Model	14
7	Optimizing the SuperNova Model	25
8	Developing the Extended SuperNova Model	28
9	Developing the Data Delivery Model	30
10	Getting Started	32
	About the Author Rick F. van der Lans	33
	About Cisco Systems, Inc.	33
	About Centennium	33

1 Introduction

The Problems of Data Vault – *Data Vault* is a modern approach for designing *enterprise data warehouses*. The two key benefits of Data Vault are *data model extensibility* and *reproducibility of reporting results*. Unfortunately, from a query and reporting point of view a Data Vault model is complex. Developing reports straight on a Data Vault-based data warehouse leads to very complex SQL statements that almost always lead to bad reporting performance. The reason is that in such a data warehouse the data is distributed over a large number of tables.

The Typical Approach to Solve the Data Vault Problems – To solve the performance problems with Data Vault, many organizations have developed countless *derived data stores* that are designed specifically to offer a much better performance. In addition, a multitude of ETL programs is developed to refresh these derived data stores periodically. Although such a solution solves the performance problems, it introduces several new ones. Valuable time must be spent on designing, optimizing, loading, and managing all these derived data stores. The existence of all these extra data stores diminishes the intended flexibility that organizations try to get by implementing Data Vault. Philosopher and statesman Francis Bacon would have said: “The remedy is worse than the disease.”

The New Approach to Solve the Data Vault Problems – An alternative approach exists that avoids the performance problems of Data Vault and preserves its key benefits: reproducibility and data model extensibility. This new approach is based on *data virtualization technology*¹ together with a data modeling technique called *SuperNova*. With this approach a data virtualization server replaces all the physical and derived data stores. So, no valuable time has to be spent on designing, optimizing, loading, and managing these derived data stores, and therefore reproducibility of reporting results and data model extensibility is retained. Also, because new derived data stores are completely virtual, they are very easy to create and maintain.

The SuperNova Modeling Technique – Applying the data modeling technique called *SuperNova* results in a set of views defined with a data virtualization server plus (very likely) one derived data store designed to support a wide range of reporting and analytical needs. SuperNova is a highly-structured modeling technique. It has been designed to be used in conjunction with Data Vault.

The Whitepaper – This whitepaper describes the SuperNova modeling technique. It explains step by step how to develop an environment that extracts data from a Data Vault and how to make that data available for reporting and analytics using the *Cisco Information Server (CIS)*. Guidelines, do’s and don’ts, and best practices are included.

Target Audience – This whitepaper is intended for those IT specialists who have experienced the problems with Data Vault as discussed in this introduction or who want to avoid these problems. In addition, general knowledge of data warehouses, star schema design, and SQL is expected.

¹ R.F. van der Lans, *Data Virtualization for Business Intelligence Systems*, Morgan Kaufmann Publishers, 2012.

2 Data Vault in a Nutshell

This section contains a short introduction to *Data Vault* and the accompanying terminology. The term Data Vault refers to two things. On one hand, it identifies an approach for designing data bases and, on the other hand, it refers to a database that has been designed according to that approach. Both are described in this section. For more extensive descriptions we refer to material by Dan Linstedt² and Hans Hultgren³.

The Data Vault – A *Data Vault* is a set of tables designed according to specific rules and meets reporting and analytical needs of an organization. The tables together generally form an *enterprise data warehouse*.

Each table of a Data Vault is normalized⁴ and according to the third normal form, but if the entire set of tables is considered, the tables are, what is called, *over-normalized* or *hyper-normalized*. Data that would normally be stored together in one table when classic normalization is applied, is distributed over many separate tables in a Data Vault. For example, an object and all its attributes are not stored together in one table, but distributed over several tables. Also, a one-to-many relationship between two objects is not implemented as a foreign key in one of the two tables, but as a separate table.

Another characteristic of Data Vault tables is that they all have particular *metadata columns* that indicate, for example, when data is loaded, from which source system the data comes, when the data was last changed, and so on. Metadata columns do not represent user information needs, but are needed to process data correctly.

The Data Vault Approach – The Other Side of the Coin – As indicated, the term Data Vault also refers to a specific modeling and design approach called the *Data Vault Approach* (in this whitepaper). It describes how information needs are to be modeled as a set of extensible tables that, together, form a Data Vault. In addition, it describes and prescribes how to unload data from data source systems, how to transform it, how to cleanse incorrect data, and how to load it into a Data Vault.

The two, the database and the approach, go hand in hand. They are two sides of the same coin. It's strongly recommended to apply the data vault approach when a Data Vault has to be designed. Also, when the approach is applied, the resulting database should represent a Data Vault.

The Tables of a Data Vault in Detail – The tables of a Data Vault are classified as *hubs*, *links*, and *satellites*. *Hubs* are tables that represent business objects, such as product, customer, airport, aircraft, and so on. *Link* tables represent relationships between hubs. Examples of link tables are flight (linking the hubs aircraft and airport) and purchase of a product (linking the hubs product and customer). *Satellites* are tables containing the attributes of hubs or links. Each satellite may contain one to many attributes. For example, a satellite of the hub airport may contain the attribute called number of runways, the employee hub may contain the monthly salary, and the customer hub may include the delivery address. An example of a link satellite is the duration of a flight belonging to the link table called flight. Hubs and links can have zero or more satellites. So, all the data stored in a Data Vault is stored in one of these three types of tables: hubs, links or satellites. Figure 1 contains an example of a data model developed for a Data Vault-based data warehouse.

² Dan E. Linstedt, *Data Vault Series 1 - Data Vault Overview*, July 2002, see <http://www.tdan.com/view-articles/5054/>

³ H. Hultgren, *Modeling the Agile Data Warehouse with Data Vault*, Brighton Hamilton, 2012.

⁴ Chris J. Date, *An Introduction to Database Systems*, 8th edition, Addison-Wesley, 2003.

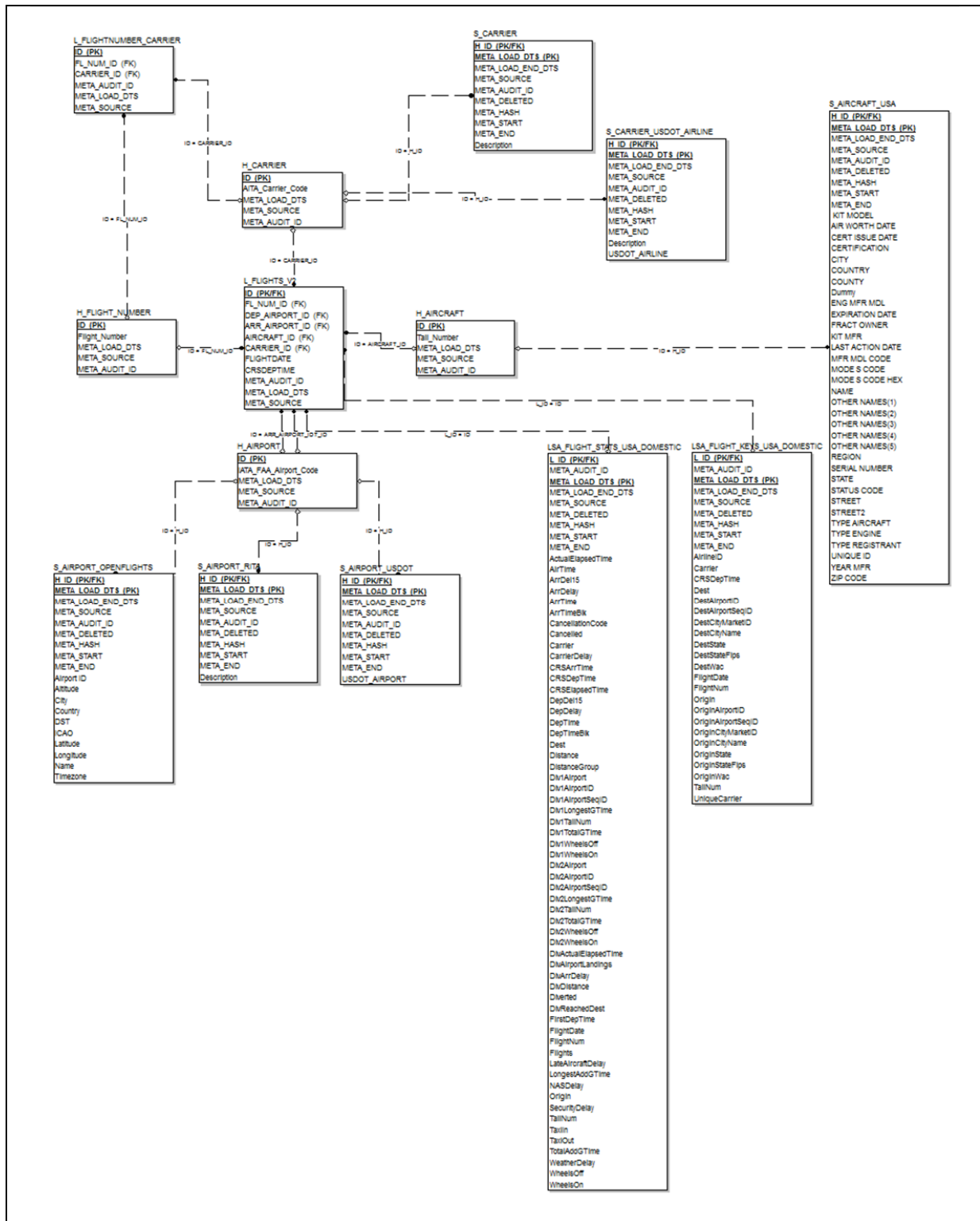


Figure 1 An example of a data vault data model.

In Figure 1, the tables whose names start with an H are hubs, with an L are links, and with an S are satellites. The tables H_AIRCRAFT, H_AIRPORT, H_CARRIER, and H_FLIGHT_NUMBER form the hubs. They are the central tables in the model. They represent important objects (or entities) within an organization. Hubs don't have many columns. There is always an identifier which is an artificial value having no meaning in real life, it's just there to represent a business object. It's what's sometimes called a *surrogate key*. In addition, each hub contains a *business key*, a value that does have value to users. It's important that this is a real unique and mandatory value. Next, there are some metadata columns, such as the date when a record was loaded in the Data Vault (META_LOAD_DTS). Note that this value may be different from the date when the object represented by the record was really created. Also, the source of where the data comes from is stored (META_SOURCE).

Satellites contain the attributes of the hubs. In this diagram H_CARRIER has two satellites, S_CARRIER and S_CARRIER_USDOT_AIRLINE, and the hub H_AIRPORT has three, S_AIRPORT_OPENFLIGHTS, S_AIRPORT_RITA, and S_AIRPORT_USDOT. Each hub table can have zero or more satellite tables. Each satellite contains the identifier of the hub and the column that indicates the date when the satellite data was loaded. In fact, a record in a satellite table represents a version of a hub's satellite data. The column META_DATA_LOAD_DTS contains a date that indicates when a record in the satellite has become obsolete. Most importantly a satellite table contains a set of real attributes of a hub. This is where most of the data resides that users need for their reporting. For example, the satellite S_AIRPORT_OPENFLIGHTS contains the attributes that indicate the longitude and latitude coordinates of an airport.

This diagram only shows one link table called L_FLIGHTS. Usually, a Data Vault contains many links. A link represents a relationship between two or more hubs. A link table has its own unique identifier, for each relationship to a hub the identifier of the hub is included as foreign key. Because most link tables represent an event, such as a money withdrawal from an ATM, a hospitalization, or a payment, they usually contain one or more date and time columns indicating when the event took place. Like hubs, link tables contain metadata columns, including the date on which the data is loaded in the Data Vault and the source system where the data is copied from.

The L_FLIGHTS link table has two satellites, LSA_FLIGHT_KEYS_USA_DOMESTIC and LSA_FLIGHT_STATS_USA_DOMESTIC. These satellite tables have the same purpose and the same structure as the satellites of the hubs except that a satellite points to a link and not to a hub.

The role of the metadata columns, such as META_LOAD_DTS, META_LOAD_END_DTS, and META_SOURCE, is explained in more detail in Section 4.

The Key Benefits of Data Vault – Other design and modeling techniques exist for developing data warehouses. For example, classic *normalization* and *star schema modeling*⁵ are well-known alternatives. All have their strong and weak points. The two strong points of Data Vault are data model extensibility and reproducibility:

- *Data model extensibility* refers to the ability to implement new information needs easily in the Data Vault. In most data warehouses, if new information needs have to be implemented, existing data structures must be changed. Those changes can lead to changes of ETL logic and to changes of reports and applications while in fact these changes may not be relevant to them at all. With Data Vault, new information leads to the introduction of new hubs, new satellites, or new links.

⁵ Wikipedia, *Star Schema*, see http://en.wikipedia.org/wiki/Star_schema, July 2014.

But never does it lead to changes of the existing data structures. They remain unaffected. So, applications and reports for which the new information needs are irrelevant, don't have to be changed. Thus a change is easier to implement. In fact, the data model extensibility property of a Data Vault makes it a highly flexible solution.

- *Reproducibility* is an aspect of *compliance*. Compliance is a state in which someone or something is in accordance with established guidelines, specifications, or legislation. One aspect of compliance is the need to reproduce reporting results or documents from the past. In a Data Vault current and historical data is organized and stored in such a way that old and new data is always available, and if the Data Vault approach is used, if a report has been created once, it can be reproduced, even if the data models have changed. To support reproducibility all the metadata columns are included in all the tables.

The Key Drawback of Data Vault – From the viewpoint of data model extensibility and reproducibility, Data Vault has a lot to offer. But from a querying standpoint a Data Vault data model is complex. Because data is distributed over many hubs, satellites, and links, developing reports straight on a Data Vault (as indicated in Figure 2) leads to very complex SQL statements in which many tables must be joined in a complex way. And complex queries are difficult to optimize by SQL database servers, often resulting in poor query performance.

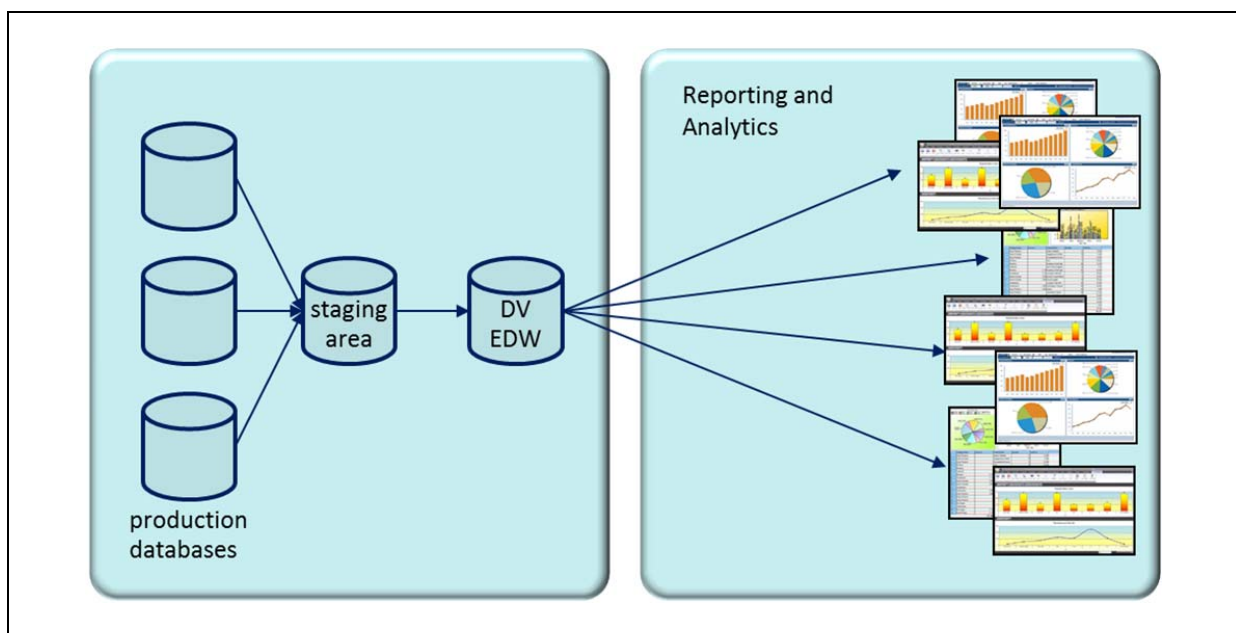


Figure 2 Running reports and queries straight on a data vault leads to complex queries and poor query performance.

For this reason, many organizations owning a Data Vault, have developed numerous *derived data stores*, such as SQL-based data marts, cubes for Microsoft SQL Server, spreadsheet files, and Microsoft Access databases; see Figure 3. These derived data stores have data structures that are more suitable for reporting and offer a much better performance.

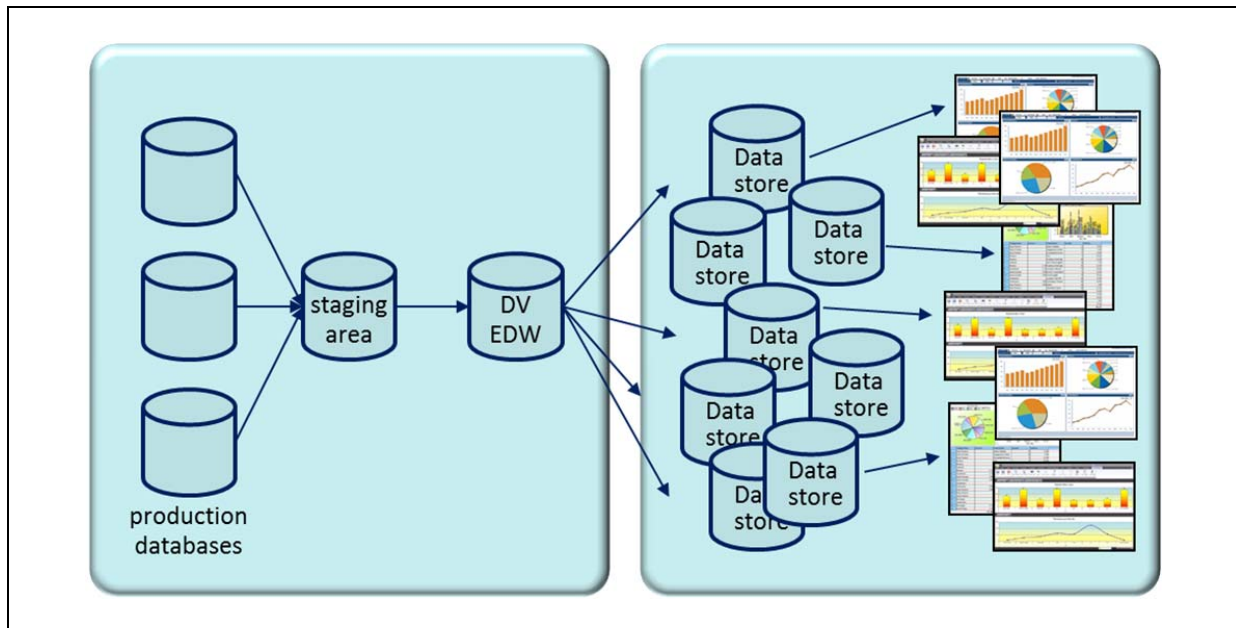


Figure 3 To improve query performance and to present easier-to-understand data structures, many derived data stores are developed.

Although such a solution solves the performance problems, it introduces some other ones:

- **Refreshing derived data stores:** All these derived data stores have to be designed, loaded with data from the Data Vault, and they have to be managed. ETL programs have to be developed and maintained to periodically refresh the data stores to keep the data up to date. They require a scheduling environment. All this work is time-consuming and costly.
- **Data model changes:** If a new satellite or link is added to a Data Vault, all the ETL programs accessing those data structures have to be changed and all the derived data stores as well. Data may have to be unloaded first, adapted ETL programs have to be executed to bring the derived data store in synch, and finally data can be reloaded back in. Some technologies used to implement derived data stores require a full recreate and full reload. All this does not improve the time-to-market. Also, all these derived data stores can end up being a maintenance nightmare.
- **No up-to-date data:** When users query a derived data store, they don't see 100% up-to-date data. There is a big chance they see outdated data, because their derived data store is refreshed, for example, only once every night.
- **Redundant data storage:** It's obvious that this solution leads to redundant storage of data across all these derived data stores.

To summarize, the intended flexibility that organizations try to achieve by implementing a Data Vault is annulled because of the need to develop many derived data stores.

Data Virtualization – The challenge for many organizations using Data Vault is to develop an environment that offers data model extensibility and reproducibility and that in addition allows for easy reporting and management. Furthermore, this environment shouldn't require the development of numerous derived data stores. Such an environment can be developed using *data virtualization*. Here, a data virtualization server replaces all the physical and derived data stores; see Figure 4. The reports see data structures that are easy to access, easy to adapt, and they do contain up-to-date data.

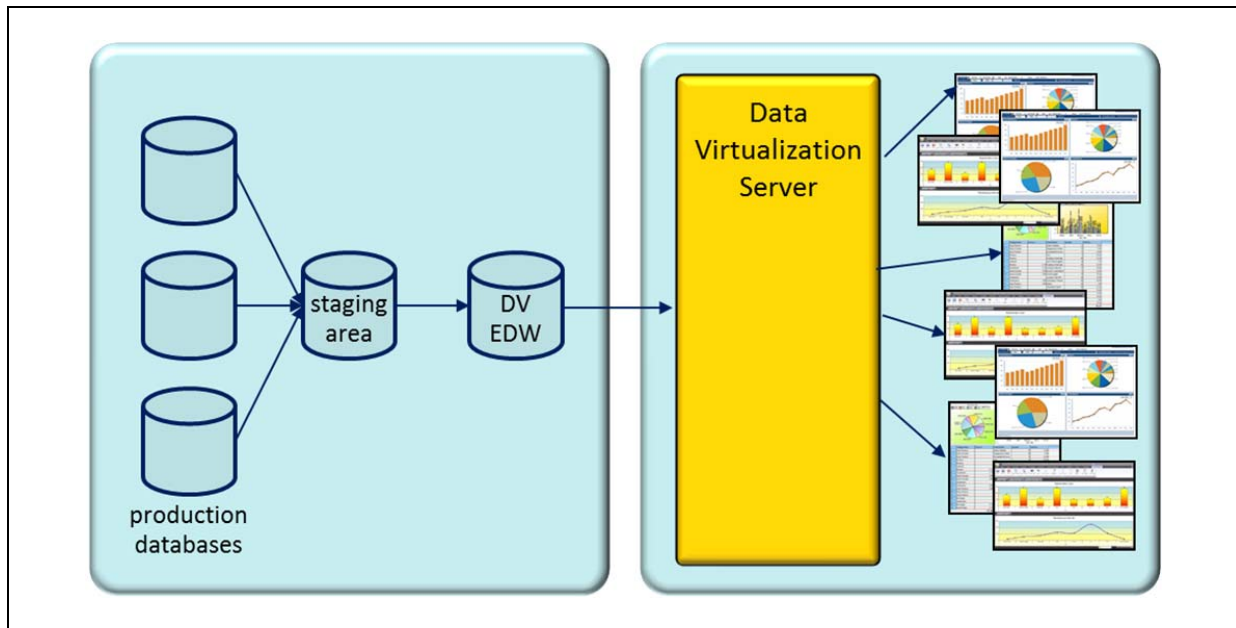


Figure 4 The data virtualization server replaces all the derived data stores.

A Short History of Data Vault – Data Vault is not a new design approach. Its history dates back to 1990. In 2000 it became a public domain modeling method. Especially between 2005 and 2010 it became well-accepted. Currently, numerous organizations have adopted Data Vault and have used it to develop their data warehouse. Because Data Vault offers a high level of reproducibility, many financial organizations were quick to see its value. Currently, acceptance is spreading. A website⁶ exists that contains a list of organizations who either use, build, or support Data Vault.

Dan Linstedt² was the founding father and is still promoting Data Vault worldwide. Hans Hultgren has also contributed much to evangelize data vault. His book³ on Data Vault definitely helped to explain Data Vault to a large audience.

As has happened with many design approaches not managed by a standardization organization, Data Vault is not always Data Vault. There are different styles. In this whitepaper one of the more common approaches is selected. But if readers use a slightly different form, they will have no problem adapting the approach to their own method and needs.

⁶ Dan Linstedt, *Data vault Users*, see <http://danlinstedt.com/about/dv-customers/#>

3 The SuperNova Data Modeling Technique

Introduction – This section introduces the data modeling technique called *SuperNova*. It has been designed to be used in conjunction with Data Vault and leans heavily on data virtualization technology. Step by step it explains how to develop an environment that extracts data from a Data Vault and makes that data available for reporting using a data virtualization server. The value of SuperNova is that the key benefits of data vault are preserved: reproducibility and data model extensibility.

The result of applying the SuperNova modeling technique results in a set of tables. These tables are implemented using the *CIS data virtualization server*. SuperNova works together with Data Vault and extends it. The goal of a SuperNova is to make the data stored in a Data Vault easily and efficiently accessible to a wide range of reports and analytics without jeopardizing reproducibility and data model extensibility and without the need to develop and manage countless derived data stores. So, SuperNova is a set of design and modeling techniques to, given an existing Data Vault, develop a layered set of data models implemented with CIS.

The Name SuperNova – The SuperNova modeling technique is named after one of the layers in the solution. It's called SuperNova because when the development of this technique started, it was known that the only workable solution would involve combining many hub, link, and satellite tables together, which would lead to large and wide tables. So, the name had to represent something big, but the word big had to be avoided, because that adjective is being overused today. Very quickly the term SuperNova was coined. A supernova is massive. Some are over 10 light years across, and a supernova remnant is even bigger: upwards of 50 million lights years. In short, something big was needed and a supernova was clearly big enough. In addition, the word nova means “new” in Latin, and new the approach certainly is.

The SuperNova Architecture – Figure 5 represents the overall architecture of a SuperNova solution. The diagram consists of five layers. The bottom layer in the diagram contains all the production databases. The layer above represents a data warehouse designed using Data Vault. The top three layers together form the SuperNova solution and are implemented using CIS.

Layer 1: Operational Systems – At the bottom of the diagram the operational systems are shown. These are the systems in which new data is entered. These can be classic data entry systems, but also brand new systems such as websites. These operational systems with their production databases fall outside the control of the data virtualization server.

Layer 2: Data Vault EDW – Data is copied from the production databases to an enterprise data warehouse designed using the Data Vault approach. This layer falls outside the control of the data virtualization server as well. The copy process and the technology used are not determined by the data virtualization server. In fact, this implementation is technologically irrelevant for the data virtualization server. SuperNova assumes that the data warehouse has been designed according to the rules of the Data Vault approach, that historical data is copied and stored in the Data Vault according to the rules, and so on. Another assumption made is that the Data Vault model is a given. So, in the solution described no changes are made or have to be made to that Data Vault. A SuperNova solution can be implemented on any existing Data Vault.

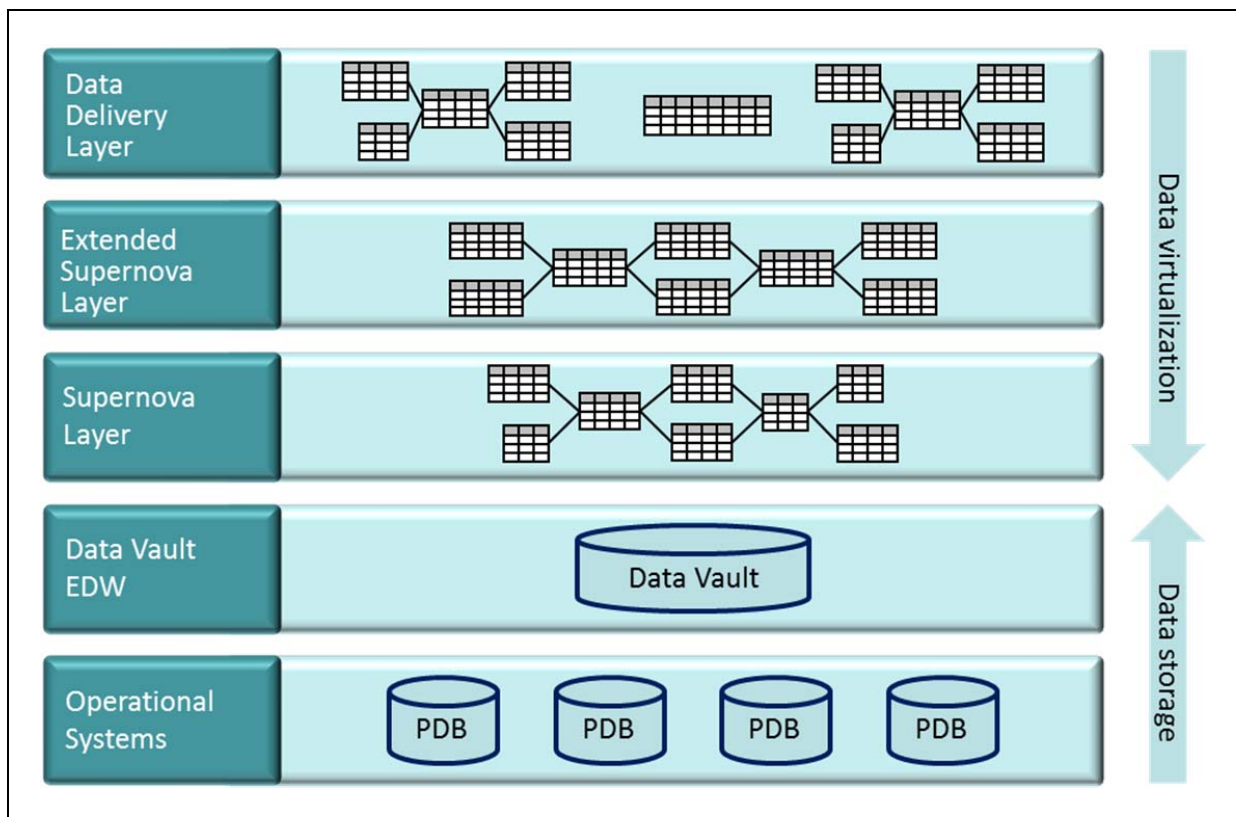


Figure 5 A SuperNova solution consists of three layers, of which the top three are managed by the data virtualization server.

Layer 3: SuperNova Layer – Layer 3, the middle layer in this diagram, is the crux of the SuperNova solution. It's a set of views defined with CIS on the Data Vault tables. In these views, data from the Data Vault tables is reorganized and restructured so that they are easy to access and that queries become straightforward. The structure of these tables form a mixture of a Data Vault model and a star schema. The tables are slightly denormalized and thus contain duplicate data.

The logic to extract data from the Data Vault tables to the SuperNova tables is relatively easy. The structure of each SuperNova table is aimed at a wide range of reporting and analytical use cases. Basically, satellite data is merged with hub data. A record in a SuperNova hub represents a version of a hub and it contains all the satellite data relevant for that specific version. The link tables are also extended with their own satellite data.

A SuperNova data model is extensible. For example, if a new satellite is added to a hub in the Data Vault layer, the corresponding SuperNova hub must be redefined, but the existing reports won't have to be changed. Or, if new links between hubs are introduced, a new SuperNova link is created. If new hubs are added, a new SuperNova hub is introduced. To summarize, the data model of the SuperNova layer is almost as extensible as that of the data vault model.

Whether these views are materialized (cached) or not depends on many factors, including the query workload, the database size, and the reporting requirements. This topic is discussed in Section 7.

Layer 4: Extended SuperNova Layer – The *Extended SuperNova* (XSN) layer consists of a set of views defined on top of the SuperNova views using CIS. In this layer the views of the SuperNova layer are extended and changed. For example, extra columns with derived data are added, column values are transformed and standardized, data is filtered, and column names are standardized. In fact, business logic is added to raise the quality level and the usefulness of the SuperNova model. The intention of this model is to contain logic that is relevant to all users, not a subset. So, all the logic implemented in the XSN layer should apply to all users and all reports.

Layer 5: Data Delivery Layer – The top layer is called the *Data Delivery* layer. This layer consists of views accessed by users and reports. For each user group or reporting environment a set of views is defined that fits their needs and works well with the technical requirements of the reporting and analytical tools. For example, some reports require tables to be organized as a star schema, while others prefer wide, flat, highly-denormalized tables. Some of the users are only interested in current data, while others need historical data as well. The structure and contents of the Data Delivery views are aimed at what the users and reports need. In a way, some of these sets of views can be qualified as *virtual data marts*.

Overall Comments – The value of the SuperNova solution is that the key benefits of Data Vault are preserved: reproducibility and data model extensibility. In addition, new derived data stores are very easy to create and maintain, because they are completely virtual. No real data stores have to be developed and managed. Although, if needed, the views of the SuperNova layer may have to be materialized for performance or scalability reasons. So, the price is storage.

The layered model approach recommended with SuperNova aligns with common data virtualization modeling best practices⁷.

4 Rules For the Data Vault Model

Designing and developing a Data Vault is a very rigid exercise. The modeling rules are very precise and clear. Still, different developers have different opinions. Therefore, there can be slight variations. These variations don't have a major impact on the SuperNova modeling technique, but can lead to minor alterations.

To avoid any confusion, here are the rules assumed in this whitepaper:

- Each hub table contains an identifier (ID) representing a hub object with a meaningless key. Each should also contain the metadata columns `BUSINESS_KEY`, `META_LOAD_DTS`, and `META_SOURCE`. The first one contains a meaningful key known by the users identifying the hub objects uniquely. The second one indicates the date and time when a hub record was loaded in the Data Vault. The third column indicates the source system (the bottom layer in Figure 5) from which the data was copied to the Data Vault. There is no column indicating the enddate, because the assumption is made that hub objects are never thrown away, not even when the business objects they represent don't exist anymore. This is required to offer reproducibility.

⁷ R. Eve, *What to Do About the Data Silo Challenge*, March 2013, see <https://roberteve1.sys-con.com/node/2591842>

- Each satellite table contains the metadata columns HUB_ID, META_LOAD_DTS, META_LOAD_END_DTS, and META_SOURCE. The first column contains the identifier of the hub to which the satellite belongs, the second column indicates the date and time when a satellite record is loaded in the table, and the third column indicates the date and time on which the record became obsolete. This is the date on which a specific set of attributes has become obsolete. The fourth column, META_SOURCE, indicates the source system from which the data is copied.
- Each hub table contains two special records. The first one identifies a so-called *Unknown* hub object. It has as identifier -1, and its business key is set to Unknown. The second record represents a so-called *Not applicable* hub record. Its identifier is set to -2 and its business key is set to N.a.. These two records are needed to link records in link tables to the hub tables even if there is no (real) hub record for that link. For both records META_LOAD_DTS is set to January 1, 1900 and the META_SOURCE is set to Repository.
- Each record in a satellite table identifies a version of a hub object. So, a specific hub object can appear multiple times in a satellite table.
- When an enddate such as META_DATA_END_DTS is unknown, not the null value but the value December 31, 9999 is inserted. This is the highest possible date in most SQL database servers. The reason is twofold. First, many conditions used in the definitions of the SuperNova views make use of the condition DATE BETWEEN STARTDATE AND ENDDATE. This condition works best if the end date has a real value and not the null value. When null would be used, the condition would look like DATE BETWEEN STARTDATE AND ENDDATE OR (DATE >= STARTDATE AND ENDDATE IS NULL). This condition is more difficult to optimize, and thus may lead to poor query performance.
- Link tables can have eventdates. These are real values. These real business dates can fall before the META_LOAD_DTS value.
- With respect to versions and historical data in the satellite tables, the following rules apply. Time-wise no gaps are allowed between two consecutive versions of a hub object. If a record does not represent the last version, its enddate is equal to the startdate of the next version + 1. Here, 1 is the smallest possible precision, such as a second or microsecond. Two versions of a hub in one satellite are *not* allowed to overlap. At a particular point in time, only one version of a hub object in a satellite table can exist. Two versions of a hub in two different satellites *are* allowed to overlap.
- With respect to data cleansing, in the Data Vault approach it's recommended not to change, correct or cleanse the data when it's copied from the production databases to the Data Vault. If data has to be cleansed, it's done after it has been loaded. In fact, the corrected data becomes the new version of the data. The incorrect, old data remains in the Data Vault just in case there is a report that was once executed on that incorrect data and must be reproduced.

5 The Sample Data Vault

In this whitepaper a small sample Data Vault is used to illustrate how SuperNova and its layers work. It consists of three hubs, one link, and five satellite tables. The first hub has two satellites, the second one only one, and the third none. The link table has two satellites. An overview of the data model of this set of tables is presented in Figure 6.

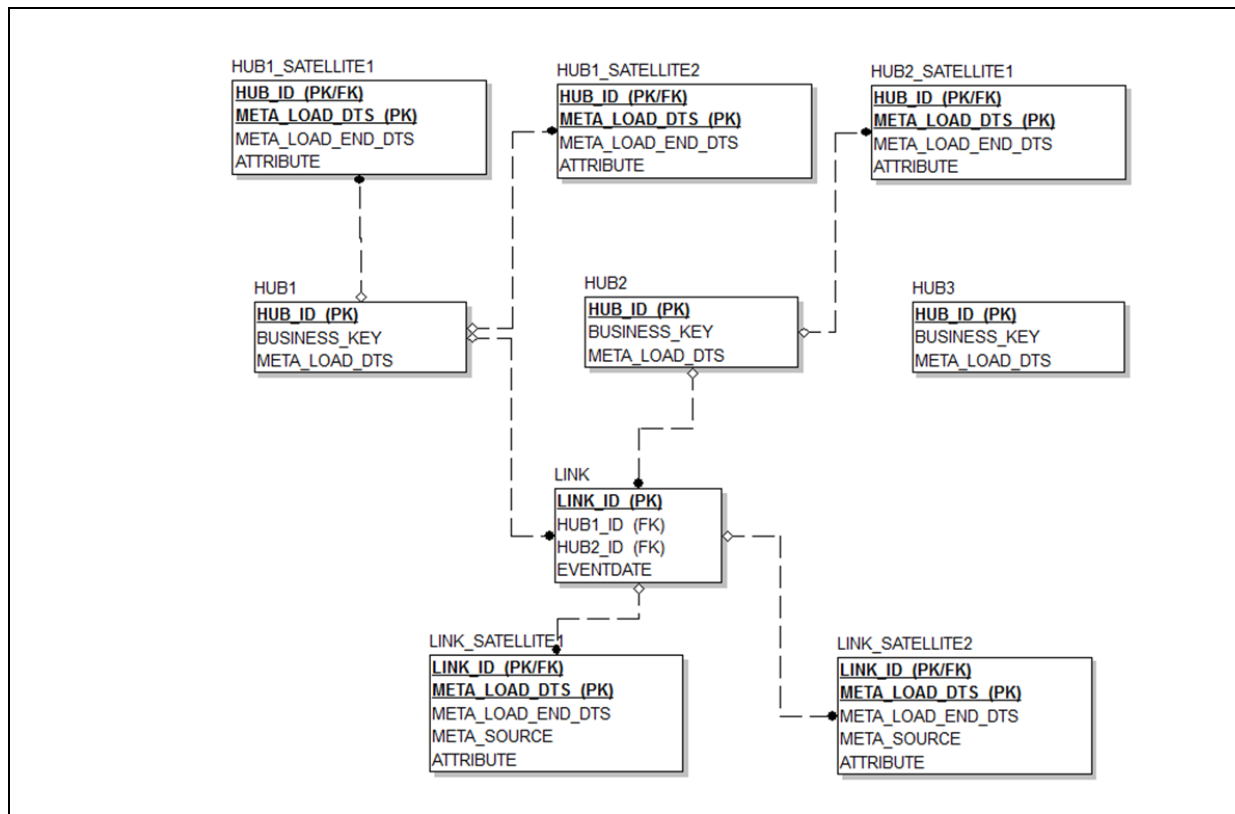


Figure 6 The data model of the sample data vault.

Next, the contents of each table is shown. On purpose, the number of rows in each table is kept small to be able to easily show what the results are of some of the steps described in the next sections.

HUB1			
HUB_ID	BUSINESS_KEY	META_LOAD_DTS	META_SOURCE
1	b1	2012-06-01 00:00:00	Demo
2	b2	2011-03-20 00:00:00	Demo
3	b3	2013-11-22 00:00:00	Demo
4	b4	2013-09-09 00:00:00	Demo
-1	Unknown	1900-01-01 00:00:00	Repository
-2	N.a.	1900-01-01 00:00:00	Repository

HUB1_SATELLITE1			
HUB_ID	META_LOAD_DTS	META_LOAD_END_DTS	ATTRIBUTE
1	2012-06-01 00:00:00	2013-11-14 23:59:59	a1
1	2013-11-15 00:00:00	2014-03-06 23:59:59	a2
1	2014-03-07 00:00:00	9999-12-31 00:00:00	a3
2	2011-03-20 00:00:00	2012-02-25 23:59:59	a4
2	2012-02-26 00:00:00	2014-02-25 23:59:59	a5
2	2014-02-26 00:00:00	9999-12-31 00:00:00	a6

HUB1_SATELLITE2			
HUB_ID	META_LOAD_DTS	META_LOAD_END_DTS	ATTRIBUTE
1	2013-06-21 00:00:00	2013-07-20 23:59:59	a7
1	2013-07-21 00:00:00	2013-11-12 23:59:59	a8
1	2013-11-13 00:00:00	9999-12-31 00:00:00	a9
2	2011-03-20 00:00:00	2012-02-25 23:59:59	a10
2	2012-02-26 00:00:00	9999-12-31 00:00:00	a11
3	2013-09-09 00:00:00	2013-11-11 23:59:59	a12
3	2013-11-12 00:00:00	2013-11-30 00:00:00	a13

HUB2			
HUB_ID	BUSINESS_KEY	META_LOAD_DTS	META_SOURCE
5	b5	2012-06-01 00:00:00	Demo
6	b6	2011-03-20 00:00:00	Demo
7	b7	2013-11-22 00:00:00	Demo
-1	Unknown	1900-01-01 00:00:00	Repository
-2	N.a.	1900-01-01 00:00:00	Repository

HUB2_SATELLITE1			
HUB_ID	META_LOAD_DTS	META_LOAD_END_DTS	ATTRIBUTE
5	2011-11-09 00:00:00	2012-07-20 23:59:59	a14
5	2012-07-21 00:00:00	2014-03-21 23:59:59	a15
5	2014-03-22 00:00:00	9999-12-31 00:00:00	a16
6	2013-12-08 00:00:00	9999-12-31 00:00:00	a17

HUB3			
HUB_ID	BUSINESS_KEY	META_LOAD_DTS	META_SOURCE
8	b8	2013-05-15 00:00:00	Demo
9	b9	2013-09-20 00:00:00	Demo
-1	Unknown	1900-01-01 00:00:00	Repository
-2	N.a.	1900-01-01 00:00:00	Repository

LINK					
LINK_ID	HUB1_ID	HUB2_ID	EVENTDATE	META_LOAD_DTS	META_SOURCE
1	1	5	2013-12-01 00:00:00	2013-12-02 00:00:00	Demo
2	1	6	2014-01-01 00:00:00	2014-01-01 00:00:00	Demo
3	2	6	2013-12-25 00:00:00	2013-12-26 00:00:00	Demo
4	3	-1	2013-06-24 00:00:00	2013-06-24 00:00:00	Demo

LINK_SATELLITE1				
LINK_ID	META_LOAD_DTS	META_LOAD_END_DTS	META_SOURCE	ATTRIBUTE
1	2013-12-01 00:00:00	2014-01-23 23:59:59	Demo	a18
1	2014-01-24 00:00:00	9999-12-31 00:00:00	Demo	a19
3	2013-12-27 00:00:00	9999-12-31 00:00:00	Demo	a20
4	2013-12-08 00:00:00	9999-12-31 00:00:00	Demo	a21

LINK_SATELLITE2				
LINK_ID	META_LOAD_DTS	META_LOAD_END_DTS	META_SOURCE	ATTRIBUTE
1	2013-12-25 00:00:00	9999-12-31 00:00:00	Demo	a22
2	2014-03-12 00:00:00	9999-12-31 00:00:00	Demo	a23
3	2014-02-02 00:00:00	9999-12-31 00:00:00	Demo	a24

6 Developing the SuperNova Model

Introduction — A SuperNova model (the middle layer in Figure 5) is derived from a Data Vault model according to a number of clear steps. A SuperNova model contains only link and hub tables. All the data of the satellite tables is merged with their respective hub or link tables. In a SuperNova model a record in a hub doesn't represent a business object, but a version of that business object. The same applies to a record in a link table. The model structure resembles that of a star schema in which fact tables are represented by links and dimensions by hubs. This section describes these steps in detail using the sample database described in Section 5. Figure 7 shows a model of the resulting SuperNova model after the design steps have been applied.

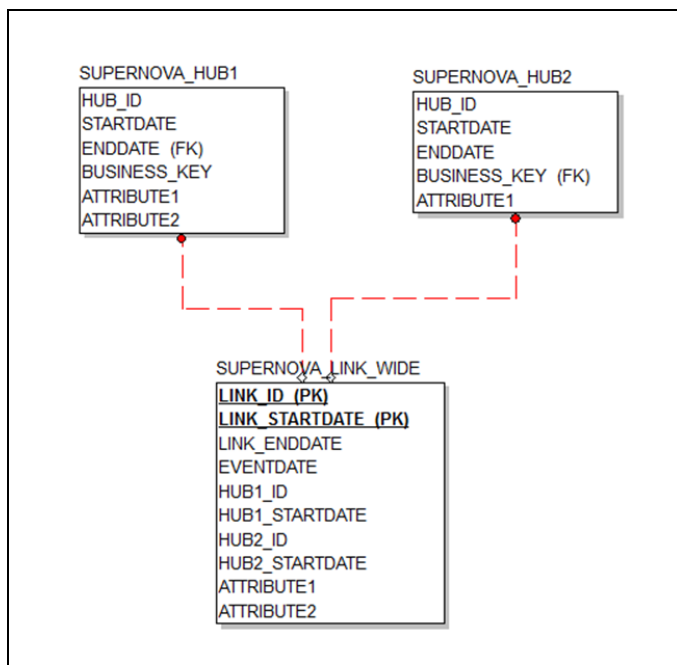


Figure 7 The SuperNova data model derived from the sample data vault.

Step 1 — Before the real SuperNova views can be defined, a layer of other views must be defined first. These views contain temporary results. For each hub table in the data vault a view is created that contains all the versions of each hub object. These *versions* are derived from the satellite tables of the hub. For example, HUB1_SATELLITE1 contains the following three records for hub object 1:

HUB_ID	META_LOAD_DTS	META_LOAD_END_DTS
1	2012-06-01 00:00:00	2013-11-14 23:59:59
1	2013-11-15 00:00:00	2014-03-06 23:59:59
1	2014-03-07 00:00:00	9999-12-31 00:00:00

and HUB1_SATELLITE2 the following three records :

HUB_ID	META_LOAD_DTS	META_LOAD_END_DTS
1	2013-06-21 00:00:00	2013-07-20 23:59:59
1	2013-07-21 00:00:00	2013-11-12 23:59:59
1	2013-11-13 00:00:00	9999-12-31 00:00:00

These records do not really represent versions of hub object 1. The real versions become visible when the two tables are merged together:

HUB_ID	STARTDATE	ENDDATE
1	2012-06-01 00:00:00	2013-06-20 23:59:59
1	2013-06-21 00:00:00	2013-07-20 23:59:59
1	2013-07-21 00:00:00	2013-11-12 23:59:59
1	2013-11-13 00:00:00	2013-11-14 23:59:59
1	2013-11-15 00:00:00	2014-03-06 23:59:59
1	2014-03-07 00:00:00	9999-12-31 00:00:00

In this result, two versions of the same hub object don't overlap. A new version ends one second before the start of a new version. If a satellite record of a hub object in HUB1_SATELLITE1 has the same startdate and enddate as one in HUB1_SATELLITE2, they form one version. The current version of a hub object has December 31, 9999 as enddate, unless an hub object is deleted. In this case, the delete date is entered as the enddate.

The diagram in Figure 8 illustrates how data from multiple satellites is merged to find all the versions. The top line indicates all the dates appearing in the startdate and enddate columns of the two satellite tables. The second line indicates the dates on which new versions for hub 1 start in satellite 1, and the third line indicates the new versions in satellite2. Underneath the dotted line the "sum" of the two is shown. It's clear that hub 1 has six versions. The first one starts at 2012-06-01, the second one at 2013-06-21, and so on. Number six is the current version and therefore has no enddate. The enddate of each version is equal to the startdate of the next version minus 1 second.

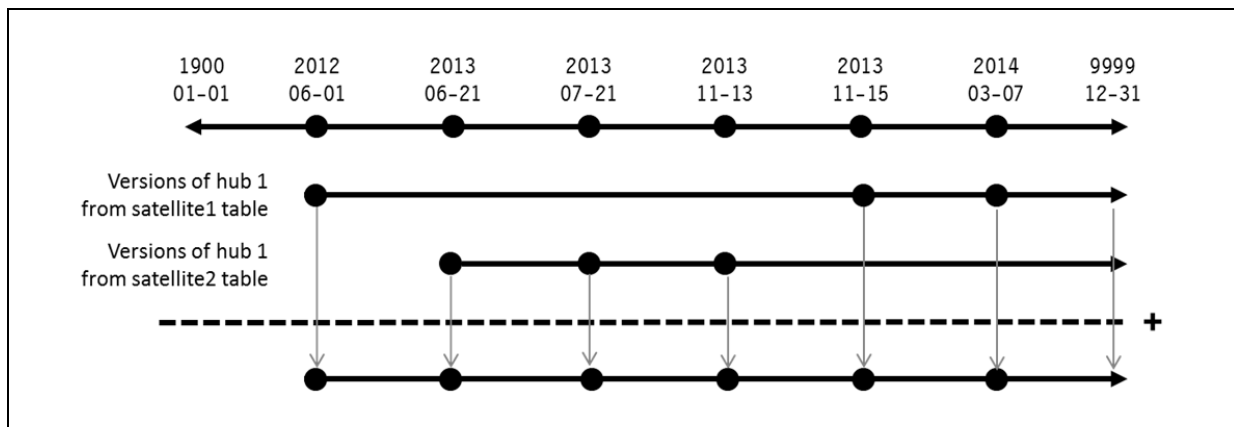


Figure 8 An illustration of all the versions of hub object 1.

The definition to create the view that contains all the versions of all objects of a hub:

```
CREATE VIEW HUB1_VERSIONS AS
WITH STARTDATES (HUB_ID, STARTDATE, ENDDATE, BUSINESS_KEY) AS (
  SELECT  HUB1.HUB_ID, SATELLITES.STARTDATE, SATELLITES.ENDDATE, HUB1.BUSINESS_KEY
  FROM    HUB1 LEFT OUTER JOIN
          (SELECT  HUB_ID, META_LOAD_DTS AS STARTDATE, META_LOAD_END_DTS AS ENDDATE
           FROM    HUB1_SATELLITE1
           UNION
           SELECT  HUB_ID, META_LOAD_DTS, META_LOAD_END_DTS
           FROM    HUB1_SATELLITE2) AS SATELLITES
          ON HUB1.HUB_ID = SATELLITES.HUB_ID)
SELECT  DISTINCT HUB_ID, STARTDATE,
        CASE WHEN ENDDATE_NEW <= ENDDATE_OLD THEN ENDDATE_NEW ELSE ENDDATE_OLD END AS ENDDATE,
        BUSINESS_KEY
FROM    (SELECT  S1.HUB_ID, ISNULL(S1.STARTDATE, '1900-01-01 00:00:00') AS STARTDATE,
                (SELECT  ISNULL(MIN(STARTDATE - '1' SECOND), '9999-12-31 00:00:00')
                 FROM    STARTDATES AS S2
                 WHERE   S1.HUB_ID = S2.HUB_ID
                 AND     S1.STARTDATE < S2.STARTDATE) AS ENDDATE_NEW,
                ISNULL(S1.ENDDATE, '9999-12-31 00:00:00') AS ENDDATE_OLD,
                S1.BUSINESS_KEY
        FROM    STARTDATES AS S1) AS S3
```

Note: This view applies the algorithm described in the previous text. In real systems, the greenish-blue terms must be replaced by the names of real tables and columns. The rest remains the same. This applies to the other SQL statements in this whitepaper as well.

Explanation: In SQL, the query in the WITH clause is called a *common table expression* (CTE). This CTE contain the following query:

```
SELECT  HUB_ID, META_LOAD_DTS AS STARTDATE, META_LOAD_END_DTS AS ENDDATE
FROM    HUB1_SATELLITE1
UNION
SELECT  HUB_ID, META_LOAD_DTS, META_LOAD_END_DTS
FROM    HUB1_SATELLITE2
```

For each satellite of the hub the values of the identifier, the startdate and enddate are selected. These results are combined using a UNION operator. The temporary result of this query is called SATELLITES. It contains the startdates and enddates off the records of all the satellites of a hub. If for a hub object multiple satellite records exist with the same startdate and enddate, the all except one are removed by the UNION operator. The (intermediate) result of this query is:

SATELLITES		
HUB_ID	STARTDATE	ENDDATE
1	2012-06-01 00:00:00	2013-11-14 23:59:59
1	2013-06-21 00:00:00	2013-07-20 23:59:59
1	2013-07-21 00:00:00	2013-11-12 23:59:59
1	2013-11-13 00:00:00	9999-12-31 00:00:00
1	2013-11-15 00:00:00	2014-03-06 23:59:59
1	2014-03-07 00:00:00	9999-12-31 00:00:00
2	2011-03-20 00:00:00	2012-02-25 23:59:59
2	2012-02-26 00:00:00	2014-02-25 23:59:59
2	2012-02-26 00:00:00	9999-12-31 00:00:00
2	2014-02-26 00:00:00	9999-12-31 00:00:00
3	2013-09-09 00:00:00	2013-11-11 00:00:00
3	2013-11-12 00:00:00	2013-11-12 00:00:00

Note that this result does not include hub object 4, because it has no satellite data.

If new satellite tables are added to a hub table, only the CTE has to be changed. It has to be extended with an extra subquery that is union-ed with the others.

In this view definition the META_LOAD_DTS and META_LOAD_END_DTS columns are used as representatives for the startdates and enddates. It could be that a satellite table also contains other types of startdates and enddates, such as the *transaction dates* (the dates on which the data was inserted in the IT system), or the *valid dates* (the dates on which the event really happened in real life). The information needs determine which set of dates is selected. There is no impact on the rest of the algorithm.

Next, the CTE is executed. Here, the original hub table itself is joined with this intermediate result called SATELLITES. The result of this CTE is called STARTDATES. A left outer join is used to guarantee that even hub objects with no satellites are included. For such objects, the startdate and enddate are set to null. The business key of the hub is included in this result as well. In fact, each column from the hub table that needs to be included in the SuperNova views, is added here. The contents of STARTDATES is:

STARTDATES			
HUB_ID	STARTDATE	ENDDATE	BUSINESS_KEY
1	2012-06-01 00:00:00	2013-06-20 23:59:59	b1
1	2013-06-21 00:00:00	2013-07-20 23:59:59	b1
1	2013-07-21 00:00:00	2013-11-12 23:59:59	b1
1	2013-11-13 00:00:00	2013-11-14 23:59:59	b1
1	2013-11-15 00:00:00	2014-03-06 23:59:59	b1
1	2014-03-07 00:00:00	9999-12-31 00:00:00	b1
2	2011-03-20 00:00:00	2012-02-25 23:59:59	b2
2	2012-02-26 00:00:00	2014-02-25 23:59:59	b2
2	2012-02-26 00:00:00	9999-12-31 00:00:00	b2
2	2014-02-26 00:00:00	9999-12-31 00:00:00	b2
3	2013-09-09 00:00:00	2013-11-11 00:00:00	b3

Table continues on the next page

3	2013-11-12 00:00:00	2013-11-12 00:00:00	b3
4	NULL	NULL	b4
-1	NULL	NULL	Unknown
-2	NULL	NULL	N.a.

There are three rows with no startdate and no enddate. The first one doesn't have them, because hub object 4 has no satellites. The other two rows have identifiers less than 0. These rows do not represent real hub objects, but are needed for the link tables; see Section 4.

This is the result of the main query of the view definition, or in other words, this is the virtual contents of the view called HUB1_VERSIONS:

HUB1_VERSIONS			
HUB_ID	STARTDATE	ENDDATE	HUB_BUSINESS_KEY
1	2012-06-01 00:00:00	2013-06-20 23:59:59	b1
1	2013-06-21 00:00:00	2013-07-20 23:59:59	b1
1	2013-07-21 00:00:00	2013-11-12 23:59:59	b1
1	2013-11-13 00:00:00	2013-11-14 23:59:59	b1
1	2013-11-15 00:00:00	2014-03-06 23:59:59	b1
1	2014-03-07 00:00:00	9999-12-31 00:00:00	b1
2	2011-03-20 00:00:00	2012-02-25 23:59:59	b2
2	2102-02-26 00:00:00	2014-02-25 23:59:59	b2
2	2014-02-26 00:00:00	9999-12-31 00:00:00	b2
3	2013-09-09 00:00:00	2013-11-11 23:59:59	b3
3	2013-11-12 00:00:00	2013-11-30 00:00:00	b3
4	1900-01-01 00:00:00	9999-12-31 00:00:00	b4
-1	1900-01-01 00:00:00	9999-12-31 00:00:00	Unknown
-2	1900-01-01 00:00:00	9999-12-31 00:00:00	N.a.

This table consists of the following four columns:

- **Hub identifier:** This column is copied from the STARTDATES result.
- **Startdate:** This value is copied from the STARTDATES result unless it is null, then the start date is set to January 1, 1900. This date represents the lowest possible date. It means that the start date of this hub version is not really known.
- **Enddate:** The end date is determined with a subquery. For each record the subquery retrieves the startdates of all the records belonging to the same hub object and determines what the oldest is of all the ones not older than the startdate of the record itself. Next, 1 second is subtracted from that startdate and that value is seen as the enddate of that version. If the startdate is null, as is the case for hubs without satellites, this version is the last version and the enddate is set to December 31, 9999. This makes it the current version for that hub object.
- **Business key:** The business key is copied from the STARTDATES result.

The above view definition is required if a hub has two or more satellites. If a hub has no satellites, the definition of its VERSIONS view looks much simpler:

```
CREATE VIEW HUB3_VERSIONS (HUB_ID, STARTDATE, ENDDATE, BUSINESS_KEY) AS
SELECT HUB_ID, ISNULL(META_LOAD_DTS, '1900-01-01 00:00:00'),
       '9999-12-31 00:00:00', BUSINESS_KEY
FROM HUB3
```

Because there are no satellites, no values for startdate and enddate exist. Therefore, they are respectively set to January 1, 1900 and December 31, 9999. This means that for each hub object one version is created, and that version is the current one.

This same view can easily be defined using the Grid of CIS; see Figure 9.

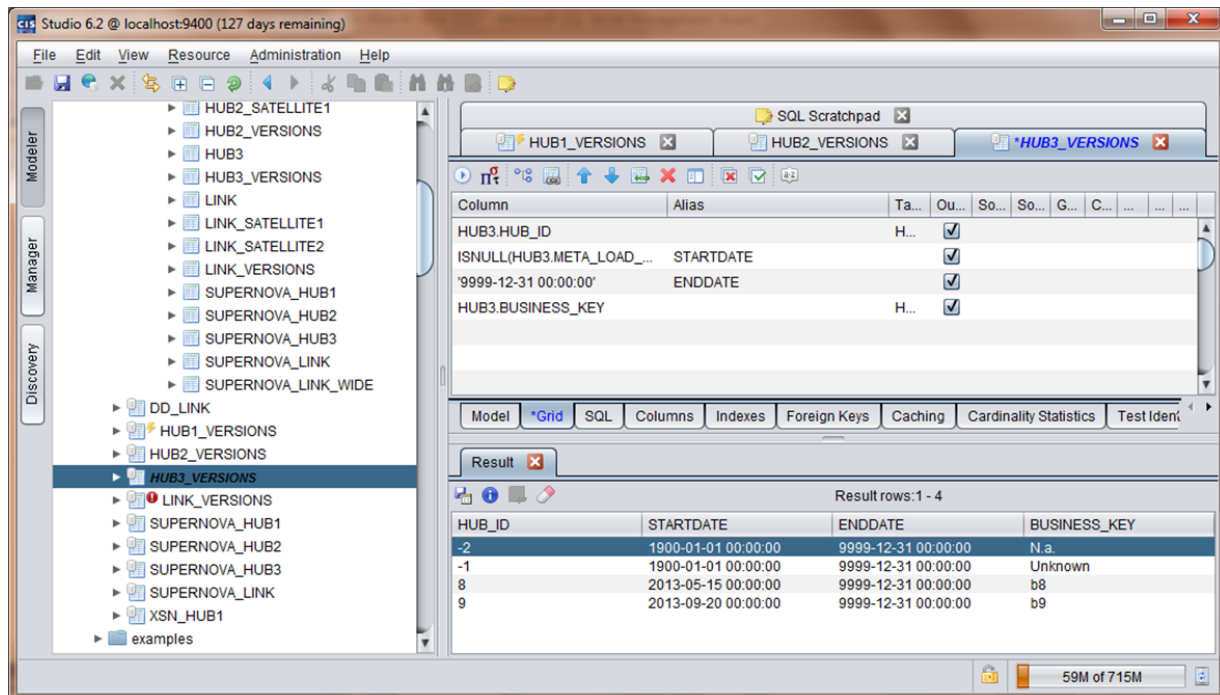


Figure 9 Defining the *HUB3_VERSIONS* view using the Grid module of CIS.

When a hub has one satellite, such as is the case with the HUB2 table, this is the preferred view definition:

```
CREATE VIEW HUB2_VERSIONS (HUB_ID, STARTDATE, ENDDATE, BUSINESS_KEY) AS
SELECT HUB2.HUB_ID, ISNULL(HUB2_SATELLITE1.META_LOAD_DTS, '1900-01-01 00:00:00'),
       ISNULL(HUB2_SATELLITE1.META_LOAD_END_DTS, '9999-12-31 00:00:00'),
       HUB2.BUSINESS_KEY
FROM   HUB2 LEFT OUTER JOIN HUB2_SATELLITE1
       ON HUB2.HUB_ID = HUB2_SATELLITE1.HUB_ID
```

The META_LOAD_DTS stored in the hub table can be used as the startdate, and the enddate is set to December 31, 9999. If the hub does contain some enddate, it's recommended to use that one.

Step 2 – Now that for each hub table in the Data Vault Model a view has been defined that shows all the versions of each hub object, the SuperNova version of each hub table can be defined. The SuperNova version contains a record for each version of a hub object plus all the attributes coming from all the satellite tables. With respect to the metadata columns, they are all left out, because they don't represent information needs.

View definition:

```
CREATE VIEW SUPERNOVA_HUB1
(HUB_ID, STARTDATE, ENDDATE, BUSINESS_KEY, ATTRIBUTE1, ATTRIBUTE2) AS
SELECT HUB1_VERSIONS.HUB_ID, HUB1_VERSIONS.STARTDATE, HUB1_VERSIONS.ENDDATE,
HUB1_VERSIONS.BUSINESS_KEY,
HUB1_SATELLITE1.ATTRIBUTE, HUB1_SATELLITE2.ATTRIBUTE
FROM HUB1_VERSIONS
LEFT OUTER JOIN HUB1_SATELLITE1
ON HUB1_VERSIONS.HUB_ID = HUB1_SATELLITE1.HUB_ID AND
(HUB1_VERSIONS.STARTDATE <= HUB1_SATELLITE1.META_LOAD_END_DTS
AND HUB1_VERSIONS.ENDDATE >= HUB1_SATELLITE1.META_LOAD_DTS)
LEFT OUTER JOIN HUB1_SATELLITE2
ON HUB1_VERSIONS.HUB_ID = HUB1_SATELLITE2.HUB_ID AND
(HUB1_VERSIONS.STARTDATE <= HUB1_SATELLITE2.META_LOAD_END_DTS
AND HUB1_VERSIONS.ENDDATE >= HUB1_SATELLITE2.META_LOAD_DTS)
```

The contents of this SUPERNOVA_HUB1 view:

SUPERNOVA_HUB1					
HUB_ID	STARTDATE	ENDDATE	HUB_BUSINESS_KEY	ATTRIBUTE1	ATTRIBUTE2
1	2012-06-01 00:00:00	2013-06-20 23:59:59	b1	a1	NULL
1	2013-06-21 00:00:00	2013-07-20 23:59:59	b1	a1	a7
1	2013-07-21 00:00:00	2013-11-12 23:59:59	b1	a1	a8
1	2013-11-13 00:00:00	2013-11-14 23:59:59	b1	a1	a9
1	2013-11-15 00:00:00	2014-03-06 23:59:59	b1	a2	a9
1	2014-03-07 00:00:00	9999-12-31 00:00:00	b1	a3	a9
2	2011-03-20 00:00:00	2012-02-25 23:59:59	b2	a4	a10
2	2102-02-26 00:00:00	2014-02-25 23:59:59	b2	a5	a11
2	2014-02-26 00:00:00	9999-12-31 00:00:00	b2	a6	a11
3	2013-09-09 00:00:00	2013-11-11 23:59:59	b3	NULL	a12
3	2013-11-12 00:00:00	2013-11-30 00:00:00	b3	NULL	a13
4	1900-01-01 00:00:00	9999-12-31 00:00:00	b4	NULL	NULL
-1	1900-01-01 00:00:00	9999-12-31 00:00:00	Unknown	NULL	NULL
-2	1900-01-01 00:00:00	9999-12-31 00:00:00	N.a.	NULL	NULL

In this view the HUB_VERSIONS view is joined with each satellite table. The complex part of this view definition is formed by the used join conditions. Here is the join condition to join HUB1 with HUB_SATELLITE1:

```
HUB1_VERSIONS.HUB_ID = HUB1_SATELLITE1.HUB_ID AND
(HUB1_VERSIONS.STARTDATE <= HUB1_SATELLITE1.META_LOAD_END_DTS
AND HUB1_VERSIONS.ENDDATE >= HUB1_SATELLITE1.META_LOAD_DTS)
```

This join condition determines to which version in the HUB_VERSIONS table a hub relates in the HUB1_SATELLITE1 table. The trick is to determine whether the startdate-enddate pair of the hub version overlaps with a satellite's startdate-enddate pair. This is the case when the startdate of the hub falls before the satellite's enddate and if the enddate of the hub falls after the satellite's startdate.

Whether all the attributes of a satellite should be included, depends on the information needs. Adding some later is a relatively simple exercise, because only the view definition has to be changed.

For the HUB2 table that has only one satellite table, the view definition is almost identical, except there is only one left outer join:

```
CREATE VIEW SUPERNOVA_HUB2
(HUB_ID, STARTDATE, ENDDATE, BUSINESS_KEY, ATTRIBUTE1) AS
SELECT HUB2_VERSIONS.HUB_ID, HUB2_VERSIONS.STARTDATE, HUB2_VERSIONS.ENDDATE,
HUB2_VERSIONS.BUSINESS_KEY, HUB2_SATELLITE1.ATTRIBUTE1
FROM HUB2_VERSIONS
LEFT OUTER JOIN HUB2_SATELLITE1
ON HUB2_VERSIONS.HUB_ID = HUB2_SATELLITE1.HUB_ID AND
(HUB2_VERSIONS.STARTDATE <= HUB2_SATELLITE1.META_LOAD_END_DTS
AND HUB2_VERSIONS.ENDDATE >= HUB2_SATELLITE1.META_LOAD_DTS)
```

For hubs without satellites, the view definition is very straightforward, because there are no attributes and thus no joins:

```
CREATE VIEW SUPERNOVA_HUB3
(HUB_ID, STARTDATE, ENDDATE, BUSINESS_KEY) AS
SELECT HUB_ID, STARTDATE, ENDDATE, BUSINESS_KEY
FROM HUB3_VERSIONS
```

Step 3 – For each link table we must go through the same process as for the hubs. For each link table in the Data Vault a view is created that contains all the versions of each link object.

```
CREATE VIEW LINK_VERSIONS AS
WITH STARTDATES (LINK_ID, STARTDATE, ENDDATE, HUB1_ID, HUB2_ID, EVENTDATE) AS (
SELECT LINK.LINK_ID, SATELLITES.STARTDATE, SATELLITES.ENDDATE,
LINK.HUB1_ID, LINK.HUB2_ID, LINK.EVENTDATE
FROM LINK LEFT OUTER JOIN
(SELECT LINK_ID, META_LOAD_DTS AS STARTDATE, META_LOAD_END_DTS AS ENDDATE
FROM LINK_SATELLITE1
UNION
SELECT LINK_ID, META_LOAD_DTS, META_LOAD_END_DTS
FROM LINK_SATELLITE2) AS SATELLITES
ON LINK.LINK_ID = SATELLITES.LINK_ID)
SELECT DISTINCT LINK_ID, STARTDATE,
CASE WHEN ENDDATE_NEW <= ENDDATE_OLD THEN ENDDATE_NEW ELSE ENDDATE_OLD END AS ENDDATE,
HUB1_ID, HUB2_ID, EVENTDATE
FROM (SELECT S1.LINK_ID, ISNULL(S1.STARTDATE, '1900-01-01') AS STARTDATE,
(SELECT ISNULL(MIN(STARTDATE - INTERVAL '1' SECOND), '9999-12-31 00:00:00')
FROM STARTDATES AS S2
WHERE S1.LINK_ID = S2.LINK_ID
AND S1.STARTDATE < S2.STARTDATE) AS ENDDATE_NEW,
ISNULL(S1.ENDDATE, '9999-12-31') AS ENDDATE_OLD,
S1.HUB1_ID, S1.HUB2_ID, S1.EVENTDATE
FROM STARTDATES AS S1) AS S3
```

Contents of the SuperNova view for the link table:

LINK_VERSIONS					
LINK_ID	STARTDATE	ENDDATE	HUB1_ID	HUB2_ID	EVENTDATE
1	2013-12-01 00:00:00	2013-12-24 23:59:59	1	5	2013-12-01
1	2013-12-25 00:00:00	2014-01-23 23:59:59	1	5	2013-12-01
1	2014-01-24 00:00:00	9999-12-31 00:00:00	1	5	2013-12-01
2	2014-03-12 00:00:00	9999-12-31 00:00:00	1	6	2014-01-01
3	2013-12-27 00:00:00	2014-02-01 23:59:59	2	6	2013-12-25
3	2014-02-02 00:00:00	9999-12-31 00:00:00	2	6	2013-12-25
4	2013-12-08 00:00:00	9999-12-31 00:00:00	3	-1	2013-06-24

Explanation: This view contains for each version of a link object a version. For each link version it shows the startdate, enddate, the eventdate, and the identifiers of the hubs to which the link version points. In this example, link 4 is linked to hub -1, which is the Unknown hub. In other words, there is no link to the HUB2 table.

Step 4 – As for the hubs, for each link table a SuperNova version must be defined that shows all the versions for each link object:

```

CREATE VIEW SUPERNOVA_LINK
(LINK_ID, HUB1_ID, HUB2_ID, STARTDATE, ENDDATE, EVENTDATE,
ATTRIBUTE1, ATTRIBUTE2) AS
SELECT LINK_VERSIONS.LINK_ID, LINK_VERSIONS.HUB1_ID, LINK_VERSIONS.HUB2_ID,
LINK_VERSIONS.STARTDATE, LINK_VERSIONS.ENDDATE,
LINK_VERSIONS.EVENTDATE,
LINK_SATELLITE1.ATTRIBUTE, LINK_SATELLITE2.ATTRIBUTE
FROM LINK_VERSIONS
LEFT OUTER JOIN LINK_SATELLITE1
ON LINK_VERSIONS.LINK_ID = LINK_SATELLITE1.LINK_ID AND
(LINK_VERSIONS.STARTDATE <= LINK_SATELLITE1.META_LOAD_END_DTS
AND LINK_VERSIONS.ENDDATE >= LINK_SATELLITE1.META_LOAD_DTS)
LEFT OUTER JOIN LINK_SATELLITE2
ON LINK_VERSIONS.LINK_ID = LINK_SATELLITE2.LINK_ID AND
(LINK_VERSIONS.STARTDATE <= LINK_SATELLITE2.META_LOAD_END_DTS
AND LINK_VERSIONS.ENDDATE >= LINK_SATELLITE2.META_LOAD_DTS)

```

Contents of the SuperNova link view (the time component is not shown in the columns STARTDATE and EVENTDATE, because they are all equal to 00:00:00):

SUPERNOVA_LINK							
LINK_ID	HUB1_ID	HUB2_ID	STARTDATE	ENDDATE	EVENTDATE	ATTRIBUTE1	ATTRIBUTE2
1	1	5	2013-12-01	2013-12-24 23:59:59	2013-12-01	a18	NULL
1	1	5	2013-12-25	2014-01-23 23:59:59	2013-12-01	a18	a22
1	1	5	2014-01-24	9999-12-31 00:00:00	2013-12-01	a19	a22
2	1	6	2014-03-12	9999-12-31 00:00:00	2014-01-01	NULL	a23
3	2	6	2013-12-27	2014-02-01 23:59:59	2013-12-25	a20	NULL
3	2	6	2014-02-02	9999-12-31 00:00:00	2013-12-25	a20	a24
4	3	-1	2013-12-08	9999-12-31 00:00:00	2013-06-24	a21	NULL

Step 5 – To make the views even easier to access, an alternative, “wider” version of the SuperNova link tables can be defined. This SuperNova link table contains the identifiers and the startdates for each hub that it is connected to. Those two columns identify a unique hub version. Note that this only works when

each link has some kind of eventdate: the date on which the link became relevant. So, this can be the departure date of a flight or the date of a purchasing transaction.

```
CREATE VIEW SUPERNOVA_LINK_WIDE
(LINK_ID, LINK_STARTDATE, LINK_ENDDATE, EVENTDATE,
HUB1_ID, HUB1_STARTDATE, HUB2_ID, HUB2_STARTDATE,
ATTRIBUTE1, ATTRIBUTE2) AS
SELECT SUPERNOVA_LINK.LINK_ID,
SUPERNOVA_LINK.STARTDATE, SUPERNOVA_LINK.ENDDATE, SUPERNOVA_LINK.EVENTDATE,
ISNULL(SUPERNOVA_HUB1.HUB_ID, -1),
ISNULL(SUPERNOVA_HUB1.STARTDATE, '1900-01-01 00:00:00'),
ISNULL(SUPERNOVA_HUB2.HUB_ID, -1),
ISNULL(SUPERNOVA_HUB2.STARTDATE, '1900-01-01 00:00:00'),
SUPERNOVA_LINK.ATTRIBUTE1, SUPERNOVA_LINK.ATTRIBUTE2
FROM SUPERNOVA_LINK
LEFT OUTER JOIN SUPERNOVA_HUB1
ON SUPERNOVA_LINK.HUB1_ID = SUPERNOVA_HUB1.HUB_ID
AND (SUPERNOVA_LINK.EVENTDATE BETWEEN
SUPERNOVA_HUB1.STARTDATE AND SUPERNOVA_HUB1.ENDDATE)
LEFT OUTER JOIN SUPERNOVA_HUB2
ON SUPERNOVA_LINK.HUB2_ID = SUPERNOVA_HUB2.HUB_ID
AND (SUPERNOVA_LINK.EVENTDATE BETWEEN
SUPERNOVA_HUB2.STARTDATE AND SUPERNOVA_HUB2.ENDDATE)
```

Contents of the wide SuperNova link view:

SUPERNOVA_LINK_WIDE									
LINK_ID	LINK_STARTDATE	LINK_ENDDATE	EVENTDATE	HUB1_ID	HUB1_STARTDATE	HUB2_ID	HUB2_STARTDATE	ATTRIBUTE1	ATTRIBUTE2
1	2013-12-01	2013-12-24 23:59:59	2013-12-01	1	2013-11-15	5	2012-07-21	a18	NULL
1	2013-12-25	2014-01-23 23:59:59	2013-12-01	1	2013-11-15	5	2012-07-21	a18	a22
1	2014-01-24	9999-12-31 00:00:00	2013-12-01	1	2013-11-15	5	2012-07-21	a19	a22
2	2014-03-12	9999-12-31 00:00:00	2014-01-01	1	2013-11-15	6	2013-12-08	NULL	a23
3	2013-12-27	2014-02-01 23:59:59	2013-12-25	2	2012-02-26	6	2013-12-08	a20	NULL
3	2014-02-02	9999-12-31 00:00:00	2013-12-25	2	2012-02-26	6	2013-12-08	a20	a24
4	2013-12-08	9999-12-31 00:00:00	2013-06-24	-1	1900-01-01	-1	1900-01-01	a21	NULL

Explanation: The SuperNova link table is joined with all its satellites. The trick here is to select the right hub version for each link object. This is done by locating the link version whose eventdate falls between the startdate and enddate of the hub version. If the eventdate of the link falls before the first version of the related hub, the hub called Unknown (with value -1) is returned, and the startdate is set to January 1, 1900. If a link object has no hub record (because the relationship is optional or not relevant), it takes the hub objects with values -1 or -2, which have the lowest possible startdate and the highest possible enddate.

There are alternative ways to deal with links whose eventdate fall before the first version of a hub. It may be useful to connect that link to the first version of the hub. In this case, the following join is required. The join condition is extended with a subquery that selects the first version of the hub:

```

LEFT OUTER JOIN SUPERNOVA_HUB1
  ON SUPERNOVA_LINK.HUB1_ID = SUPERNOVA_HUB1.HUB_ID
  AND (SUPERNOVA_LINK.EVENTDATE BETWEEN
    SUPERNOVA_HUB1.STARTDATE AND SUPERNOVA_HUB1.ENDDATE) OR
    SUPERNOVA_LINK.EVENTDATE <
    (SELECT MIN(SUPERNOVA_HUB1.STARTDATE)
     FROM SUPERNOVA_HUB1
     WHERE SUPERNOVA_LINK.HUB1_ID = SUPERNOVA_HUB1.HUB_ID)

```

There are two advantages of using wide views. First, because the wide version contains many columns, users and reports will have to do less joins. This can have a positive impact on query performance. Second, it allows for foreign keys to be defined between the link views and its hubs, which is required by some reporting tools.

As an overview, Figure 10 presents all the other views and tables on which the SUPERNOVA_LINK_WIDE view is dependent.

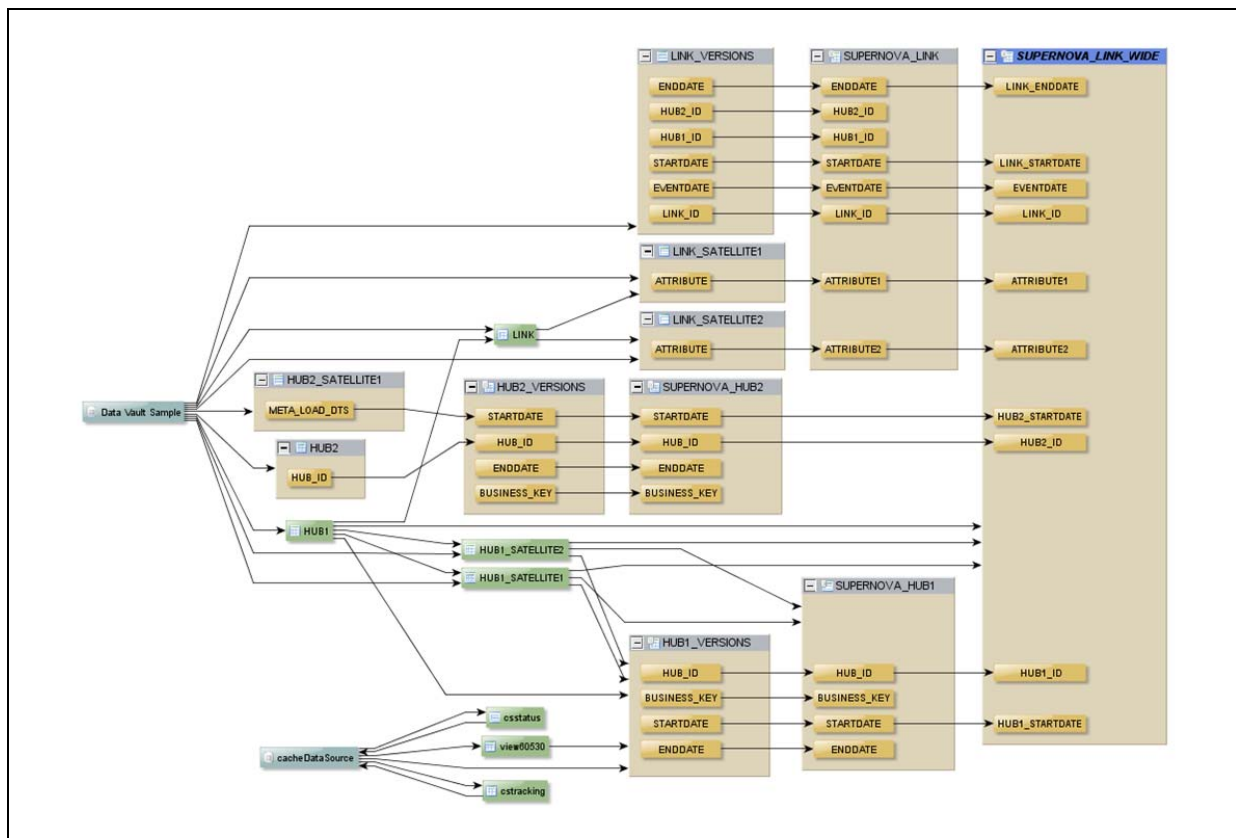


Figure 10 An overview of views and tables on which the SUPERNOVA_LINK_WIDE view is dependent.

Step 6 — Define primary keys on the SuperNova hub and link views. Because they are views, they don't do anything. Nevertheless, it's recommended to define them, because many reporting and analytical tools need to know what the primary key of each table is. By defining these keys, that metadata is exposed and can be retrieved by the tools. The tools will understand the structure of the hubs and links better and they can more intelligently generate queries. Figure 11 shows how the primary key for the SUPERNOVA_HUB1 view is defined.

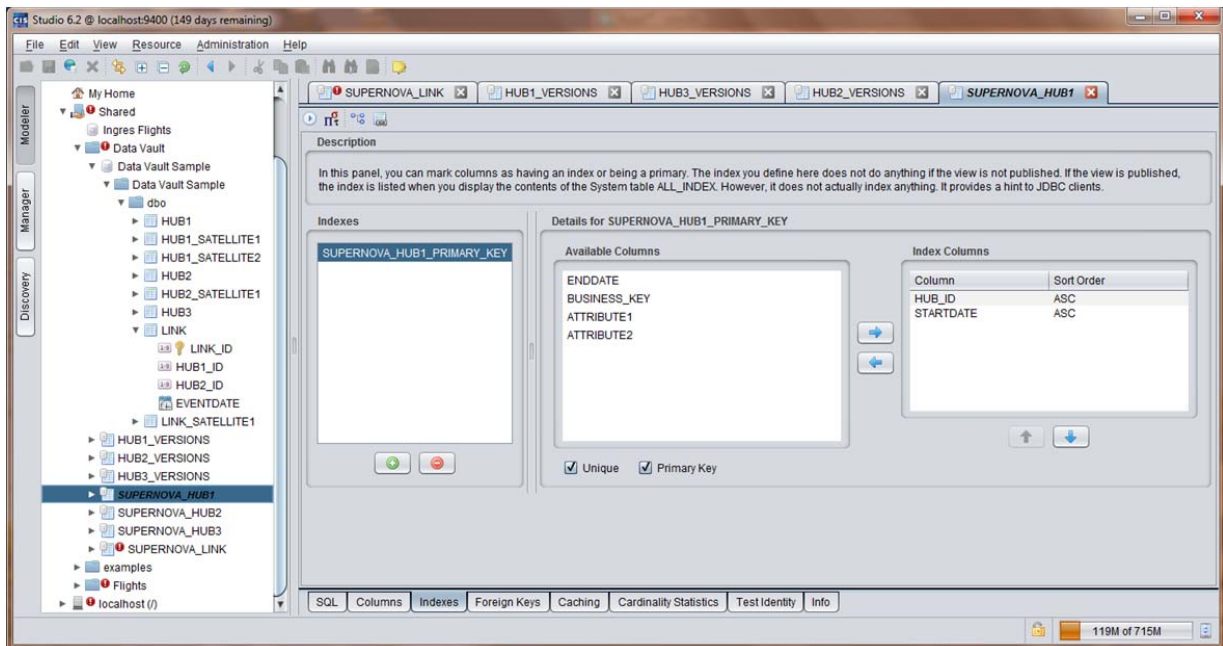


Figure 11 Definition in CIS of the primary key of the SUPERNOVA_HUB1 view.

Step 7 – What applies to primary keys, applies to foreign keys as well. Again, the reason is that many reporting and analytical need to know the relationships (foreign keys) between the views. As example, here is the SQL code to define two foreign keys from the SUPERNOVA_LINK_WIDE view to the two hub views.

```
ALTER TABLE SUPERNOVA_LINK_WIDE
  ADD FOREIGN KEY (HUB1_ID, HUB1_STARTDATE) REFERENCES
    SUPERNOVA_HUB1 (HUB_ID, STARTDATE)

ALTER TABLE SUPERNOVA_LINK_WIDE
  ADD FOREIGN KEY (HUB2_ID, HUB2_STARTDATE) REFERENCES
    SUPERNOVA_HUB2 (HUB_ID, STARTDATE)
```

Figure 12 shows how the foreign key linking the SUPERNOVA_LINK_WIDE view to the SUPERNOVA_HUB1 view is defined.

Note that foreign keys cannot be defined for the original SuperNova link tables. The reason is that the primary key of each hub consists of the hub identifier and the startdate. And it's the startdate that's missing from the link table.

7 Optimizing the SuperNova Model

The performance of queries on the views of the SuperNova layer is dependent on many aspects, including the hardware being deployed, the database server in use, whether caching is switched on, how much data the views contain, and the query workload. This section contains some recommendations for optimizing the SuperNova model when performance problems are encountered. Note that these are very generic recommendations.

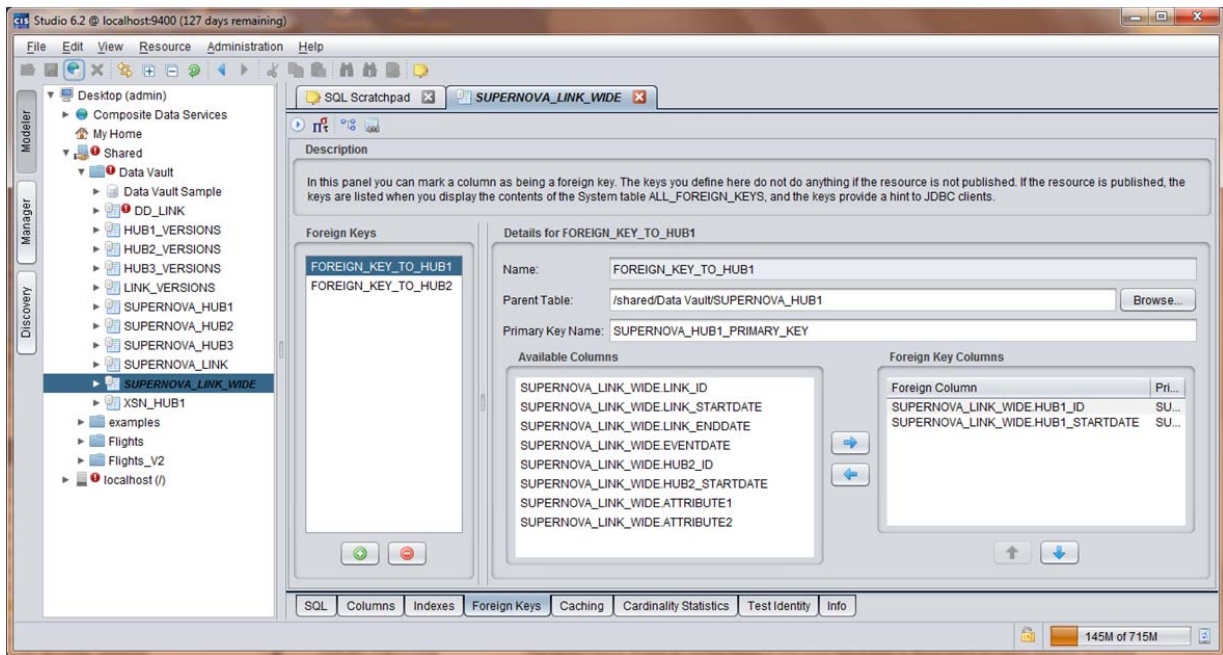


Figure 12 Define in CIS what the foreign key of a SuperNova view is.

Updating Statistical Information Of Data Vault Tables — A classic reason why performance may be poor is that the statistical information on the tables and indexes in the Data Vault are out of date. They don't reflect the correct situation anymore. Because the query optimizer of a SQL database server bases its processing strategy on that statistical information, it's crucial that it's always up to date. How statistical information can be updated is product-dependent. We refer to the manuals.

Caching of SuperNova Views — For those SuperNova views (links and hubs) that exhibit a poor query performance, define a cache. Figure 13 contains a screenshot that indicates how a cache is defined on the view SUPERNOVA_HUB1.

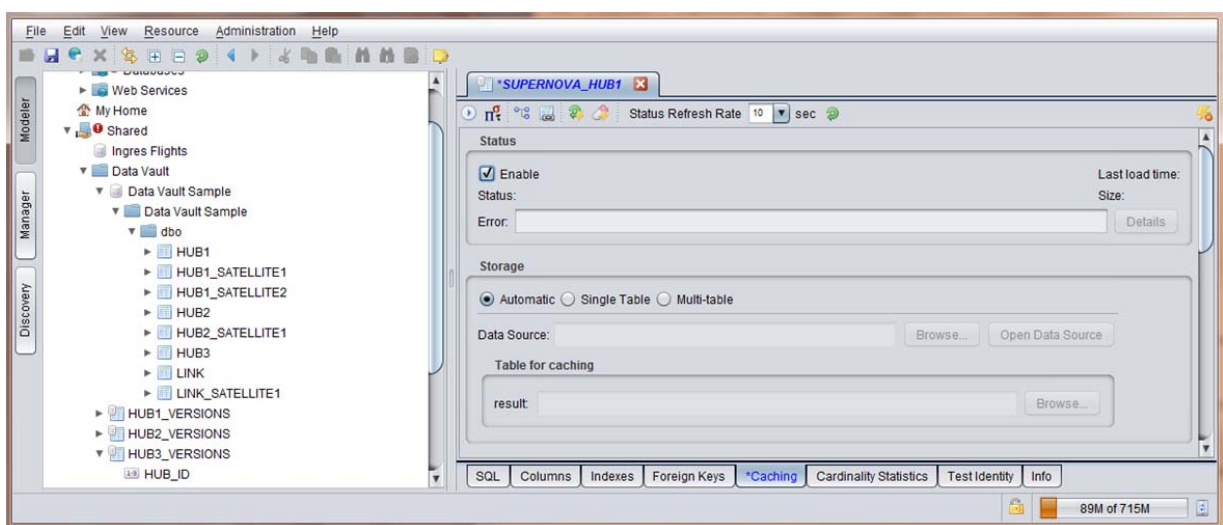


Figure 13 Define a cache for a SuperNova view.

It's recommended that these caches are stored in a SQL database server. This has as an advantage that extra indexes can be defined which speed up performance even more.

For each cached view, a refresh schedule must be defined. An example of a simple refresh schedule is: every 24 hours at night using a full refresh. The refresh schedule depends on the requirements concerning the actuality of data. With respect to the order of refreshes, first refresh all the SuperNova hub views and then the SuperNova link views.

If performance on a view is still unacceptable, for example because it contains millions of rows and the queries continuously accesses a large portion of the rows, it's recommended to create a cache for the VERSIONS view on which the view itself is defined. If that still not sufficient, it's worth trying to turn the STARTDATES CTE into a cached view as well.

Updating Statistical Information Of Caches — As it's important that statistical information for Data Vault tables is up to date, it's also important that statistical information on the cached views is also up to date. Be sure that after every refresh of a cache, an update of the statistical information is considered. Out-of-date statistical information can lead to poor query performance.

Indexes on Cached Views — By default, CIS implements cached views with files. Access to those files can be fast, but it's not the fastest data storage technology available, especially if files become large. Therefore, CIS allows caches to be stored in SQL database servers.

The advantage of using a SQL database server is that all its performance improving features can be deployed to speed up access to the caches, such as memory buffer optimization, table partitioning, and tablespace organization. One of those features is that extra indexes can be defined on the caches to speed up query performance.

Defining indexes on the foreign keys in the SuperNova link tables makes sense. Also, define indexes on columns that are aggregated often. The same applies for columns on which many filters are applied. Note that adding extra indexes can slow down refreshing of the caches somewhat.

Developing the indexes on the caches can't be done from within CIS. This must be done using the SQL database server itself. When an extra index is defined, be sure that the right parameters are set; see the manuals of the SQL database server.

Note: The structure of a table used for caching a view is a little different from the view itself. CIS adds one column called CACHEKEY for internal processing. It contains an identifier of the refresh run. A separate table indicates what that identifier means. Never change anything with respect to this column. Nor, should the indexes that are defined by CIS itself be dropped.

Optimizing the SQL Database Server — Each SQL database server offers a wide range of features to improve performance. Tables can be partitioned, buffers can be enlarged and tuned, table space parameters can be optimized, data can be moved to faster storage technology, and so on. Check with database administrators to see if they can tune and optimize access to the cached views.

Different Database Server — CIS allows caches to be stored in SQL database servers, such as EMC Greenplum, HP Vertica, IBM DB2 LUW, IBM PureData for Analytics (Netezza), Microsoft SQL Server, MySQL, Oracle, PostgreSQL, SAP Sybase ASE, Sybase IQ, and Teradata. Most of these are much more scalable and high-

performing than files and will definitely speed up the queries on cached views.

8 Developing the Extended SuperNova Model

The views defined in the SuperNova layer present the data stored in the Data Vault but with a different structure, one that is easier to use. But the layer doesn't change the data nor does it filter out data, except the metadata is left out. It's primarily a re-arrangement of the data. In the next layer, called the *Extended SuperNova* layer (XSN), extra columns are added and values are transformed. This XSN layer consists of a set of views defined on top of the SuperNova views. Each view in the XSN layer relates to one view in the SuperNova layer (and vice versa). All the columns of a SuperNova view are included in the corresponding XSN view. What's different is that these XSN views can be extended with columns that contain derived data and values may be transformed.

Adding Derived Data — An XSN view may include columns with derived data. For example, when birth dates are available in a SuperNova hub for employees and users want to see the employee's age as well, an extra column can be defined that shows that age. Or, when a SuperNova view shows temperatures in Fahrenheit, the XSN view contains the temperature in Celsius as well.

As an example, the next view is defined on the SUPERNOVA_HUB1 view and contains two derived columns. The first one shows the number of days between the STARTDATE and the ENDDATE columns, and the second one the date of the Sunday following the STARTDATE column.

```
CREATE VIEW XSN_HUB1 AS
SELECT HUB_ID, STARTDATE, ENDDATE, BUSINESS_KEY, ATTRIBUTE1, ATTRIBUTE2,
CASE WHEN STARTDATE <> '1900-01-01 00:00:00' AND
      ENDDATE <> '9999-12-31 00:00:00'
      THEN CAST(DAYS_BETWEEN(ENDDATE, STARTDATE) AS INTEGER)
      ELSE NULL END AS DAYS_BETWEEN,
CASE WHEN STARTDATE <> '9999-12-31 00:00:00'
      THEN CASE DAYOFWEEK(STARTDATE)
            WHEN 1 THEN STARTDATE + INTERVAL '7' DAY
            WHEN 2 THEN STARTDATE + INTERVAL '6' DAY
            WHEN 3 THEN STARTDATE + INTERVAL '5' DAY
            WHEN 4 THEN STARTDATE + INTERVAL '4' DAY
            WHEN 5 THEN STARTDATE + INTERVAL '3' DAY
            WHEN 6 THEN STARTDATE + INTERVAL '2' DAY
            WHEN 7 THEN STARTDATE + INTERVAL '1' DAY
            END
      ELSE STARTDATE END AS NEXT_SUNDAY
FROM SUPERNOVA_HUB1
```

Explanation: The formula to derive the number of days in between is a little bit more complex than expected. The reason is that we must cater for the situation that the startdate is January 1, 1900 and the enddate is December 31, 9999. In both cases, the number of days in between is set to null. The formula for determining the next Sunday after a particular data is also a little tricky.

Contents of the XSN_HUB1 view:

XSN1_HUB1							
HUB_ID	STARTDATE	ENDDATE	HUB BUSINESS KEY	ATTRIBUTE1	ATTRIBUTE2	DAYS BETWEEN	NEXT SUNDAY
1	2012-06-01	2013-07-20 23:59:59	b1	a1	a7	NULL	1900-01-07
1	2013-07-21	2013-11-12 23:59:59	b1	a1	a8	NULL	1900-01-07
1	2013-11-13	2013-11-14 23:59:59	b1	a1	a9	415	2012-06-03
1	2013-11-15	2014-03-06 23:59:59	b1	a2	a9	115	2013-07-28
1	2014-03-07	9999-12-31 00:00:00	b1	a3	a9	2	2013-11-17
2	2011-03-20	2012-02-25 23:59:59	b2	a4	a10	112	2013-11-17
2	2102-02-26	2014-02-25 23:59:59	b2	a5	a11	NULL	2014-03-09
2	2014-02-26	9999-12-31 00:00:00	b2	a6	a11	343	2011-03-27
3	2013-09-09	2013-11-11 23:59:59	b3	NULL	a12	731	2012-03-04
3	2013-11-12	2013-11-30 00:00:00	b3	NULL	a13	NULL	2014-03-02
4	1900-01-01	9999-12-31 00:00:00	b4	NULL	NULL	64	2013-09-15
-1	1900-01-01	9999-12-31 00:00:00	Unknown	NULL	NULL	18	2013-11-17
-2	1900-01-01	9999-12-31 00:00:00	N.a.	NULL	NULL	NULL	1900-01-07

Note: Include derived columns for which the formula requires only access to values stored in the same row. So, don't include aggregated data, move those to the next layer, the Data Delivery layer.

Transforming Data — The second purpose of the XSN views is transforming data. The definition of an XSN view can include logic to transform values. For example, a column containing employee genders may contain different codes indicating female, in the view definition these codes may be transformed to the correct code. Or, if two values in a Data Vault column have been concatenated as one, it may be necessary to split them as two separate values/columns. It may also be necessary to transform certain columns in such a way that it becomes easier for users to integrate the data from multiple tables.

To develop more complex transformations, it may be necessary to define extra tables that, for example, contain lists of incorrect values and their correct counterparts. These tables can be defined in the same database where the views are cached, so that a join of the cached views and code tables can be pushed down by CIS to that database server.

Reuse of Definitions — Defining XSN views has several practical advantages. First, by defining derived columns and by transforming values at this layer, all the users share the same definitions regardless of the reporting tool they use. This leads to more consistent reporting results. Second, it saves development time. The logic has to be developed only once. Third, some of the simpler reporting tools don't offer the functionality to define derived columns or do complex transformation, which makes this a useful alternative.

Always XSN — Even if there are no extra columns and even if there are no transformations needed, it's strongly recommended to define XSN views on top of the SuperNova views. If the XSN views exist, it becomes easier to hide changes to the SuperNova views that are irrelevant for certain users or reports. In other words, if the Data Vault tables change and the SuperNova views change accordingly, those changes can be hidden for those users and reports not interested in the new data.

9 Developing the Data Delivery Model

The top layer in Figure 5 is called the *Data Delivery layer* (DD). DD views are defined on top of the XSN views. For each user group a separate set of DD views is defined; see Figure 14. DD views show the data in the form the users and reports need, or the form which it is most convenient for them. With respect to their virtual contents DD views can overlap.

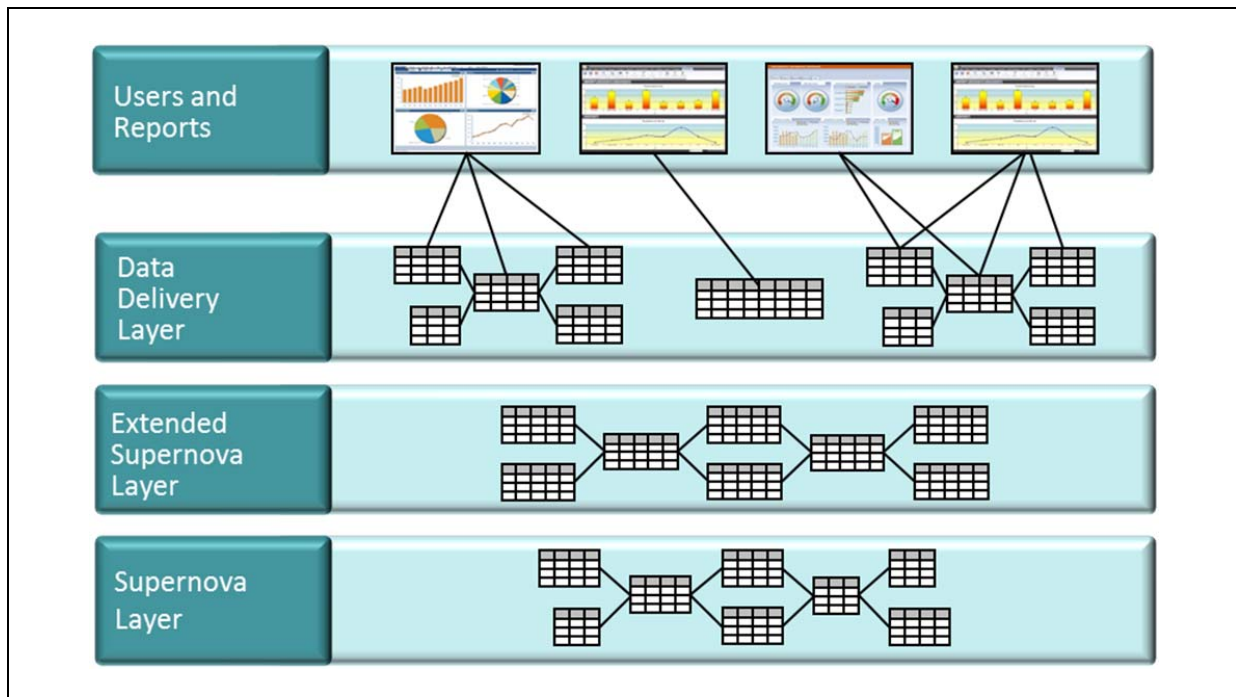


Figure 14 *User groups have different sets of DD views.*

As examples, five different forms of DD views are described here:

- Data is shown in a filtered manner
- Data is shown in aggregated form
- Data is shown in one large, highly denormalized table
- Data is shown in a star schema form
- Data is shown with a service interface

A Filtered View – A particular group of users may not be interested in all the data of an XSN view, or they may not be allowed to access all the data of a view. For example, they may only want to work with sales data for a particular sales region, or they may only see current data and not historical data. In such cases, filters are added to the DD views defined on the XSN views.

An Aggregated View – Especially link tables may contain massive amounts of records showing data on a very high level of detail. It may be more useful for users to aggregate the data somewhat. For example, purchases may be aggregated per hour, and deliveries per week or per product group. In this case, group by-operators are added to the DD views defined on the XSN views.

A Denormalized View – More and more tools for reporting and analytics prefer to have the data organized as one wide table containing all the relevant columns. Especially many of the newer self-service data visualization tools prefer this. They read all that data in the memory of the client machine, turn it internally into some fast indexed structure, and it's then presented in a user-friendly form.

In most cases, such a wide table is a denormalized version of a SuperNova link table, or in fact a denormalization of an extended SuperNova link table. This link table is joined with all relevant extended SuperNova tables:

```
CREATE VIEW DD_LINK AS
SELECT LINK_ID, HUB1_ID, HUB1_STARTDATE, SUPERNOVA_HUB1.ENDDATE AS HUB1_ENDDATE,
       HUB2_ID, HUB2_STARTDATE, SUPERNOVA_HUB2.ENDDATE AS HUB2_ENDDATE, EVENTDATE,
       SUPERNOVA_HUB1.BUSINESS_KEY AS HUB1_BUSINESS_KEY,
       SUPERNOVA_HUB1.ATTRIBUTE1 AS HUB1_ATTRIBUTE1,
       SUPERNOVA_HUB1.ATTRIBUTE2 AS HUB1_ATTRIBUTE2,
       SUPERNOVA_HUB2.BUSINESS_KEY AS HUB2_BUSINESS_KEY,
       SUPERNOVA_HUB2.ATTRIBUTE1 AS HUB2_ATTRIBUTE1
FROM SUPERNOVA_LINK_WIDE
LEFT OUTER JOIN SUPERNOVA_HUB1
  ON SUPERNOVA_LINK_WIDE.HUB1_ID = SUPERNOVA_HUB1.HUB_ID AND
   SUPERNOVA_LINK_WIDE.HUB1_STARTDATE = SUPERNOVA_HUB1.STARTDATE
LEFT OUTER JOIN SUPERNOVA_HUB2
  ON SUPERNOVA_LINK_WIDE.HUB2_ID = SUPERNOVA_HUB2.HUB_ID AND
   SUPERNOVA_LINK_WIDE.HUB2_STARTDATE = SUPERNOVA_HUB2.STARTDATE
```

Evidently, such a link table can be set up differently. For example, it now contains all the startdates and enddates. These can be removed to make the table easier to understand for users.

A Star Schema View – The structure of the SuperNova link and hub tables form a star schema pattern, where the links are *facts* and the hubs the *dimensions*. It may be necessary to simplify this structure by removing columns that are irrelevant for the users, or by showing in the hubs only current versions.

It may also be useful to combine multiple links together to form one dimension. For example, it may be useful for users to have the links sales per product and number products returned to be combined in one dimension. Careful that when link tables are joined together that the respective keys match.

A Service View – Most tools use the SQL interface when accessing data. But there are BI applications that access data through a service interface using SOAP or REST. It's recommended to determine first which interfaces they prefer and then determine the structure of a view that can support that. Next, that view is defined on the existing (SN views. By accessing these, no logic is bypassed and there is a greater chance that the results obtained via the service interface are in synch with those obtained via the SQL interface.

A Final Remark – It's not the intention to create a set of views for each report separately. The focus should be on reuse. Try to implement all the logic in the SuperNova views or the XSN views and as little in the Data Delivery layer. Use the latter for combining and restructuring purposes.

10 Getting Started

How to get started with SuperNova depends on whether an existing Data Vault already exists or a new one is being developed (green field situation).

Migrating an Existing Data Vault Environment — If a Data Vault-based data warehouse environment consisting of many derived data stores, it's recommended to select an approach where the existing derived data stores are gradually replaced by views defined in CIS. The following migration process is proposed:

- Install CIS and connect it to the Data Vault.
- Select a report that must be migrated first (depending on user priorities).
- Select the tables in the derived data stores required by that report.
- Define the SuperNova, Extended SuperNova, and the Data Delivery views to recreate the tables of the derived data stores.
- If needed, define caches for some of these views.
- Redirect the report to the Data Delivery views and test whether the results are consistent with the previous solution.
- Extract logic from the reports and move that to the Extended SuperNova views or the Data Delivery views.
- When all works fine, migrate the next report.

A Green Field Situation — In a green field situation deploy a step-by-step, iterative process as well:

- Install CIS and connect it to the database that will contain the Data Vault.
- Design your first set of Data Vault tables and apply the same rules as specified in Section 4.
- Define SuperNova views and Extended SuperNova views for all the Data Vault tables.
- Select a report that must be implemented first (depending on user priorities).
- Define the required Data Delivery views for this report, and implement as much of the logic in the views and not in the reports. If logic can be reused by other reports, implement it in the Extended SuperNova views.
- If needed, define caches for some of the views.
- When all works fine, implement the next report.

About the Author Rick F. van der Lans

Rick F. van der Lans is an independent analyst, consultant, author, and lecturer specializing in data warehousing, business intelligence, database technology, and data virtualization. He works for R20/Consultancy (www.r20.nl), a consultancy company he founded in 1987.

Rick is chairman of the annual European Enterprise Data and Business Intelligence Conference (organized annually in London). He writes for the SearchBusinessAnalytics.Techtarget.com, B-eye-Network.com⁸ and other websites. He introduced the business intelligence architecture called the *Data Delivery Platform* in 2009 in a number of articles⁹ all published at BeyeNetwork.com. The Data Delivery Platform is an architecture based on data virtualization.

He has written several books on SQL. Published in 1987, his popular *Introduction to SQL*¹⁰ was the first English book on the market devoted entirely to SQL. After more than twenty five years, this book is still being sold, and has been translated in several languages, including Chinese, German, and Italian. His latest book¹¹ *Data Virtualization for Business Intelligence Systems* was published in 2012.

For more information please visit www.r20.nl, or email to rick@r20.nl. You can also get in touch with him via LinkedIn and via Twitter @Rick_vanderlans.

About Cisco Systems, Inc.

Cisco (NASDAQ: CSCO) is the worldwide leader in IT that helps companies seize the opportunities of tomorrow by proving that amazing things can happen when you connect the previously unconnected. Cisco Information Server is agile data virtualization software that makes it easy for companies to access business data across the network as if it were in a single place.

About Centennium

Centennium specializes in Business Intelligence and Data Warehousing and is a leader in Data Vault based Business Intelligence solutions. Centennium is the Dutch partner of Genesee Academy and provides various Data Vault classes and Data Vault Certification CDVDM.

⁸ See <http://www.b-eye-network.com/channels/5087/articles/>

⁹ See <http://www.b-eye-network.com/channels/5087/view/12495>

¹⁰ R.F. van der Lans, *Introduction to SQL; Mastering the Relational Database Language*, fourth edition, Addison-Wesley, 2007.

¹¹ R.F. van der Lans, *Data Virtualization for Business Intelligence Systems*, Morgan Kaufmann Publishers, 2012.