

Model Driven Data Warehousing

Stop Building Data Warehouses ...
Use them!

Version 1.1

Dr. H. (Harm) van der Lek



Audience:

This white paper is intended for people who want to understand how the design and implementation of Data Warehouses and Data Marts can be improved dramatically by using Model Driven Methods and Tools. So it concerns: IT-managers, Consultants, and Designers in the BI-arena. Even interested Business Managers can profit from these insights. It explains the principles and shows how they are implemented in the Model Driven Data Warehouse Generator BIReady™.

Contents

1	Introduction	3
1.1	The evolution of OLAP tools.....	3
1.2	The evolution of Data Warehousing	3
1.3	Business processes and Business Intelligence.....	4
2	Building a Data Warehouse	4
2.1	Traditional Data Warehousing.....	5
2.2	Inflexibility Causes.....	5
2.3	The ETL bottleneck	6
3	Generating a Data Warehouse.....	8
3.1	Generic Architecture.....	8
3.2	What is a 'business model'?	9
3.3	Suitable views	10
3.4	Staging format	13
3.5	Inside the transparent black box.....	14
4	History of Data	16
4.1	History of administration	16
4.2	History Types.....	17
5	An Example: BIReady™	18
5.1	Business Model.....	18
5.2	Data Mart Wizard.....	19
5.3	Dimension and Facts Edit.....	20
5.4	Generated Star schema	21
5.5	Mapping Screen.....	22
5.6	Process Instructions	23
5.7	Starting the process.....	24
5.8	Log Report of Run.....	25
6	Conclusions	26

1 Introduction

This document is based on the experience of the author that he is (and others are) doing the same things over and over in Data Warehouse projects. We all are re-inventing not only other people's wheels but even our own wheels. This applies to both the design and the implementation issues.

First of all we remark that a Data Warehouse is not a goal on itself. It is, however, inevitable. In many circumstances it is simply a good architecture to transfer data to a special environment, if one wants to have a Business Intelligence Environment.

Building or generating a Data Warehouse Environment, that is the question. Hence chapters 2 and 3. In chapter 4 we illustrate one of the most striking 'generic' issues: The history of data. A real world example as in chapter 5 is probably the best way to illustrate matters. Of course in the last chapter we will draw conclusions and wrap up the paper.

1.1 The evolution of OLAP tools

In fact what we want to point out is not new as a phenomenon in the history of automation. People not only automated business processes, like financial administration, but they automated the automation itself as well. In the BI field this is illustrated by the evolution of the OLAP tools. First every organization that wanted a 'management information' application, implemented computer programs themselves with, what we call today, 'drill-down' functionality. But then it turned out that this functionality was 'generic', which means that it was (on a certain level of abstraction) always the same. But this means that it could be programmed only once! That is why vendors started to build and sell OLAP tools.

If we compare the situation of OLAP tools with the phenomenon of 'designing and building' a Data Warehouse, we conclude that most organizations are in the stage of 'building themselves' or 'hiring (expensive) subcontractors to do so'. But can it be compared? Only if in the field of Data Warehousing there are also 'generic' issues. Issues that can be solved by a piece of software. And there are! Examples: storing history of data, surrogate warehouse keys, logging, etc. Even the design of the Data Warehouse and the Data Marts (as databases) can be generated. The only thing that is needed is the so-called 'business model'. This is why this approach is called 'Model Driven'. In this paper we will look for these generic aspects and explain how this generative software will work.

1.2 The evolution of Data Warehousing

Data Warehousing emerged from the fact that companies wished to have access to enterprise data to produce valuable information. Since it was not possible and/or desirable to query source data directly, one started to make a copy of the data. This 'copy management' turned out to have serious disadvantages. First of all, since the data was continuously changing, one had to repeat the copy process. These 'multiple copies' lead to unacceptable large amounts of data. Even worse: An analysis of historical data was hard if not impossible (queries had to run over the various copies). Furthermore the structure of the data was equal to the original structure, which was optimized for

operational use and not for querying. So other design paradigms came to the front, especially star join schemas. This means that the structure of source and target differs. This fact is stressed by ETL-tool vendors, because it makes their software unavoidable. So nowadays it seems that source and target are hardly related at all. And of course this is not true either, because they are! We may hope that both are about the same business.

1.3 Business processes and Business Intelligence

In the following figures we illustrate the last remark ("it is about the same business"). On the one hand we have a group of people involved in the operational processes (producing, selling, transporting, building, etc. See Figure 1). In modern organizations



Figure 1 The business itself ...



Figure 2 ... and the BI environment

these processes are supported by IT systems and data. On the other hand we have the executives and knowledge workers who want to manage the organization and they need business information to do so (See Figure 2). The structure to offer this information is the 'BI Environment'. Now, hopefully, both groups are slaving away in the same business! Otherwise the organization doesn't need a Data Warehouse, but an organization consultant instead.

This observation will lead to an important assumption, namely that the so-called business staging area (See paragraph 3.4) can be populated from the operational systems.

2 Building a Data Warehouse

In this chapter we analyze the current situation in many Data Warehouse projects and environments. It is generally recognized, that the design of the Data Warehouse (as database) and the implementation of the ETL processes is the most critical and risky part of a Business Intelligence project (even if we do not consider the problems of dirty data, which in most cases have to be solved by a culture change of the organization anyway). So if we can gain time and increase the quality here it is worth to do so.

2.1 Traditional Data Warehousing

Indeed various researchers have reported that most Data Warehouse projects are not successful. These projects take too much time, exceed budget and, even worse, do not meet expectations. In Figure 3 we show the architecture of a traditional BI Environment. Central is the 'Enterprise Data Warehouse'. It holds data from various sources, and supposed to contain 'the single version of the truth'. The Data Marts differ from each other as these are meant to be used by specific user groups. However, they are consistent with each other because they are derived from the Enterprise DWH. For example, the total turnover in one month according to the Data Mart of the production manager should be the same as the total turnover in the same month according to the Data Mart of the marketing manager. Otherwise, there is no reliable information manage the organization. There is some discussion between gurus (Ralph Kimball versus Bill Inmon) about the design of this enterprise Data Warehouse, but almost everybody agrees that star schema design is a good idea for the Data Marts.

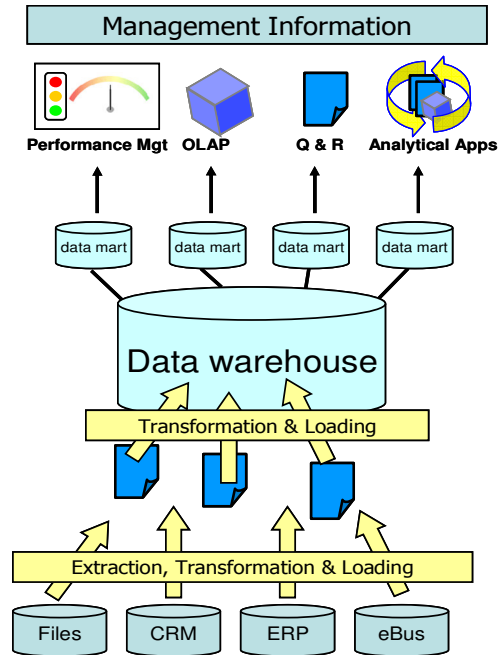


Figure 3 Traditional DWH Architecture

2.2 Inflexibility Causes

Despite the fact that the architecture of the last figure is not bad in itself, it turns out to be hard to do a good job designing and implementing it. Especially if one wants to have a 'flexible' environment. Let us analyze why this is so. First of all we have to define what we mean by flexibility: We call a BI architecture flexible if we can adapt it to changing business requirements without excessive cost. The latter also implies that it can be changed without loss of existing data. In our opinion there are two main causes for inflexibility: (1) Strong dependency of the various parts of the architecture and (2) building up history. The first cause is illustrated in Figure 4. Some vendors try to solve the first problem by providing 'Meta Data Tools'. The idea is that if one can analyze the impact of a change by having access to all the Meta data of the various parts, then it is possible to determine the consequences of the change and to do the necessary adaptations, without surprises that some parts do not function anymore.

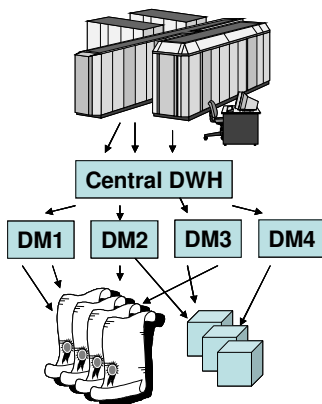


Figure 4 Strong inter dependency

We think that the role of Meta data is indeed very important, but – and this is a very crucial clue! – it should be the other way around: starting from a relatively small

amount of Meta data, which are defined upfront, the whole environment should be generated! This approach is proven to be possible.

Figure 6 illustrates the second reason why Data Warehouse environments, especially star schema designs, tend to be inflexible. Suppose we have a table where we keep history. Usually this is done by adding a new record if something changes (stacking). The 'slowly changing dimension technique' (Kimball type 2) is an example of this. But if we think of one of the simplest ways we could change the structure of this database (and not lose the existing data) is to add a column to this table. Technically this is not hard: ALTER TABLE ... ADD COLUMN But the problem is: what are the values that one can or should input to this new column.

Product Key	Product Code	Start Date	End Date	Courant	Packing
1	A30j	10	40	Yes	Box
2	A30j	40	50	Yes	Crate
3	A30j	50	9999	No	Crate
4	B00	20	9999	Yes	Bottle

Figure 6 Storing history by stacking

Product Code	Start Date	End Date	Courant	Packing	Production Location
A30j	10	20	Yes	Box	Amsterdam
A30j	20	40	Yes	Box	Bombay
A30j	40	50	Yes	Crate	Bombay
A30j	50	9999	No	Crate	Bombay
B00	20	60	Yes	Bottle	Amsterdam
B00	60	9999	Yes	Bottle	Hong Kong

Figure 5 History table after adding a column

Suppose we want to add 'Production location' and we still know that A30j was produced in Amsterdam and thereafter (as from date 20) in Bombay. B00 at first also in Amsterdam, as from 60 in Hong Kong. Then, if we think a while of the situation, we conclude that our table should look like Figure 5. So in general it is even necessary to add new rows. Not to think about the consequences for the (large!) fact table involved.

2.3 The ETL bottleneck

If the Data Warehouse project often the most troublesome part of a Business Intelligence project, then within this project the ETL is by far the most risky task.

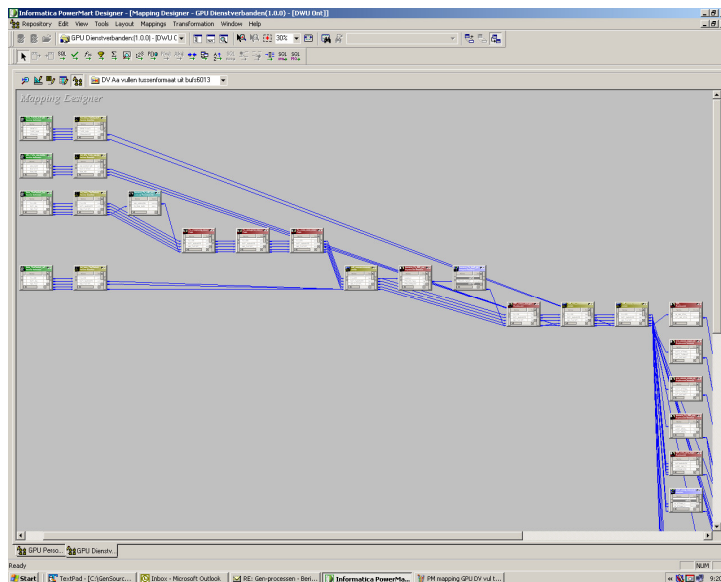


Figure 7 (too?!) complicated mapping

No programming?

What about the claim of the ETL tool vendors, that life will be easy if you use their product? After all they claim that you do not have to 'program' anymore. But if you look at an real example as shown, you can ask yourself if this is programming or not. It sure turns out to be as 'buggy' as normal programming. Surely, it is a visual way of working, but where classic programming means 'writing lines of code', this means

'drawing lines of code'. The conclusion is: it still is not 'Model Driven'.

E, T or L? (ETTL, ELT)

If we think a moment of the future of ETL, we can suspect that the 'E' of extraction will stay. It will always be necessary to fetch your source data one way or the other. One aspect of the E, namely the so called CDC (Changed Data Capture) will become even more important. It turns out to be a 'generic' problem, but if you are faced with huge amounts of (legacy) data which are slowly changing (e.g. you have 10 million customers) then you probably need very specialized software to fetch only the changes after the last load in a performant way. But the rest (T and L) can surely be solved by a generic (Model Driven) approach. This means that vendors are able to make appropriate software to solve these issues. But here is another important clue: this can only be achieved, if you do not restrict yourself to the ETL arena only! One should take into account the design and the implementation of the Data Warehouse itself and generate this by a Model Driven approach as well. In the next section we will illustrate this point.

Model driven architecture for ETL?

Lets us assume for a moment that we are an ETL software vendor and that we want to develop a 'Model Driven' or 'Meta Data Driven' approach. This means that we have to ask ourselves what this Meta data should be. The only answer can be: the Meta data of Source and Target. But this task turns out to be too complex and/or it results in the ETL tools as they are today. Furthermore, one still has to design the targets: the Data Warehouse and/or Data marts. Which means: taking a class to learn how to do so, etc. Trying to realize Meta Data driven ETL is an example of 'tunnel vision'. Instead one should 'think out of the box' and then one gets the extricating 'Eureka': Generate the targets as well!

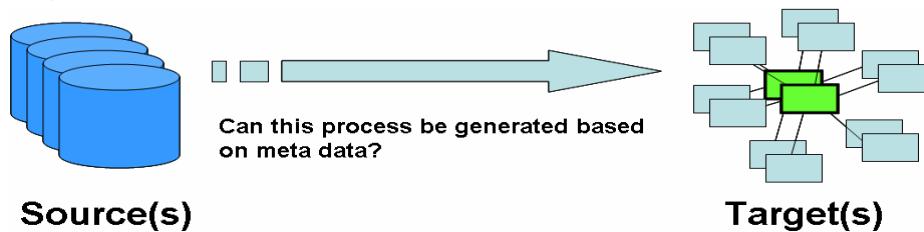


Figure 8 Model driven ETL?

The Path of the Data

Let us look at this conclusion from another angle. We think of the path the data has to go on its journey from source to target. Probably all kinds of things need to happen. Extraction of the Data and maybe advanced Changed Data Capture as noted earlier. Some custom transformations should be in place: transformations which are very specific for the situation of the company at hand. Cleansing is also an issue that can be very specific. It should be noted however, that if the quality of the source data is very poor, it is probably better to work on other IT projects (or even on a 'business culture improvement' project). A lot of transformations, cleansing (e.g. handling of referential integrity problems) and other functionalities are 'generic', which means that they can be handled by an appropriate tool. However, as argued in the forgoing paragraph, this can only be successful if one extends this MDA (Model Driven Approach or Architecture) to the field of designing and implementing (hence 'generating') the Data Warehouse and the Data marts.

3 Generating a Data Warehouse

In this chapter we will dive into the ideas and architecture of a Model Driven Approach to Data Warehousing ('Generic Data Warehousing'). We will show what the differences *and* similarities are with the classic approach. It turns out that it can be characterized as 'generating' a Data Warehouse environment instead of designing and building it.

3.1 Generic Architecture

In Figure 3 we showed the picture of the classic architecture. In fact it will turn out that the generic architecture is not that different at all. The crucial difference lies in the way it is achieved. Let us first note what the absolute constants in this picture are: (1) the fact that the users want to be able to look at the information obtainable from the data of the

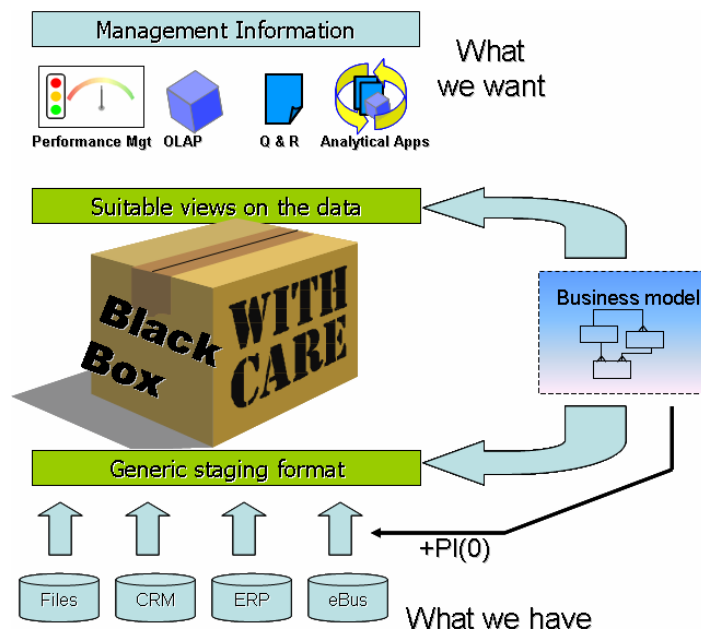


Figure 9 Generic Architecture

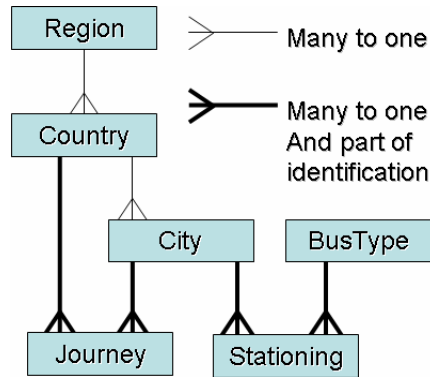
staging tables are generated based on the business model, the rest of the functionality can also be generated! In fact everything between these two layers can be considered to be a black box where the functionality is generated by a software tool provided by a vendor. In paragraph 3.5 we will show how this is done with BIReady™.

Furthermore the 'business staging' tables have to be populated with data from the (different) source systems. This can be relatively easy because as we will see (see paragraph 3.3) there are no constraints here. That means that we need some ETL (or probably only EL), but this turns out to be much more simple than in the classical projects.

company (and maybe from external data as well) and to analyze it. They want to use, for example, OLAP tools to analyze, dashboard tools for the top management, etc. And (2) data residing in the operational systems. We already noted (see Figure 1 and Figure 2) that we may assume that the users and the operational systems are involved in the same business. This means that it must be possible to make a model of that business and to derive two things from it: (a) Suitable views on the data (views that can be queried easily by users and/or tools, see paragraph 3.3) and (b) the structure of staging tables in which the data of the source systems can be copied. Now the good news: since both these business views and the

3.2 What is a 'business model'?

We spoke of 'the business model'. What do we mean by that? In fact it is a normal



information model, describing the structure of the information which is going on in a business or which describes it. The most common way to express such an information model is EAR or ER modeling. It is important that this model describes the business as it is at one moment in time. In other words: one does not have to model history! (which makes the modeling task much easier). Later on one has to add some Meta information to the model saying what history should be kept in what way. In Figure 10 a concrete example is drawn. This model is simple on the one hand, but complex enough to illustrate some important issues. It is about a travel business where they organize bus journeys from a city to a country. The bus departs from a city and drives into a country. In that country it drops passengers in all kind of resorts. For the moment it is not important what

Figure 10 A Business Model (ERD)

resorts there are, so they are not (yet!) in the model. In the ERD we only have information on the entity types and the relationships (and partly about the identification of entities by the bold relationships). To complete the model we add (the rest of) the identifications and the attributes:

Entity Type	Identification	Other attributes
Journey	Day, FromCity (from relationship), ToCountry (from relationship)	NrOfPax
Stationing	BusType (from relationship), CityCode (from relationship)	NumberStationed
BusType	BusTypeCode	Capacity, Brand
City	CityCode	CityName, CitySize
Country	CountryCode	CountryName, Polity
Region	RegionCode	RegionName, ThirdWorldYesOrNo

To illustrate the entity type 'Journey' we give a typical example

"On the 12th of July 2006 a bus departed from Los Angeles to Mexico with 40 passengers on board."

An example of 'Stationing' is:

"In San Fransisco normally 5 busses of the type LU are stationed."

Finally we remark that we restrict ourselves to models where we do not have many-to-many relationships. As is well known this is not a real restriction, since by introducing a new entity type, one can split a many-to-many relationship in two many-to-one relationships. This means that each relationship has a *direction*. The relationship between

'Country' and 'Region' in Figure 10, for example, is understood to point from 'Country' to 'Region'.

3.3 Suitable views

With the bus example of paragraph 3.2 in mind we can make clear what we mean by 'Suitable views' and why they can be derived (uniquely!) from the business model. Before doing this we consider a typical question that a user would like to ask:

Question A: "How many passengers traveled from a City in Germany to a country in the third world?"

It should be clear that this question can be answered based on information that is structured according to the model in paragraph 3.2. After all a country lies in a region and from this region we know if it is a third world region or not, by using the appropriate attribute of region. To answer the question one needs a 'view' that contains (at least) the following attributes: FROM_COUNTRYCODE, TO_THIRDWORLDYESORNO, NROFPAX. Then the following SQL code will do the job:

```
select sum(NROFPAX)
from view
where FROM_COUNTRYCODE = 'GE'
and TO_THIRDWORLDYESORNO = True
```

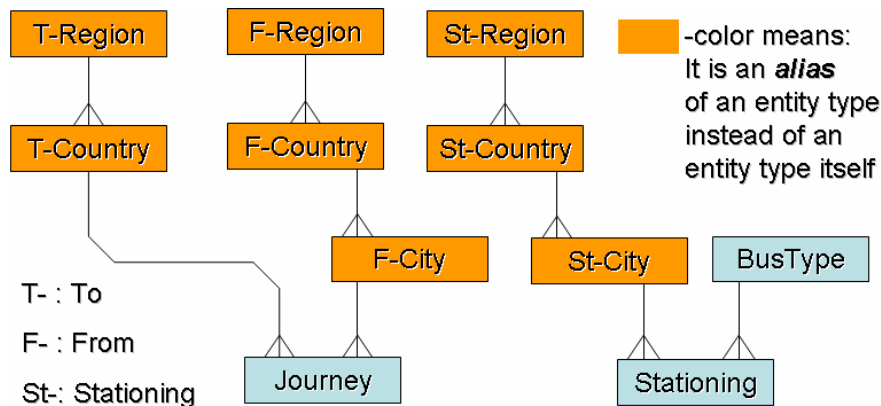


Figure 11 An unfolded form of the business model

Now, can such a view be derived from our model? First we change our ERD from Figure 10 in the following way. We acknowledge that the entity type 'City' can play two roles: a city can be a city where busses are stationed (a St-City) and/or it can be a city where a bus departs (a from-City or F-City). We call this 'aliases'. This *aliasing* is a well known technique. The same way we conclude that we have the following: Aliases of Country: To-Country, From-Country, Stationing-Country. Aliases of Region: To-Region, From-Region, Stationing-Region. This way we get an unfolded form of the business model in Figure 11.

Now if we consider this diagram we observe that in fact we have a 'model' consisting of two disconnected parts, both of which have the structure of a *snowflake* (in graph theory it is called a tree, and since there can be one or more trees, we call Figure 11 a *forest*).

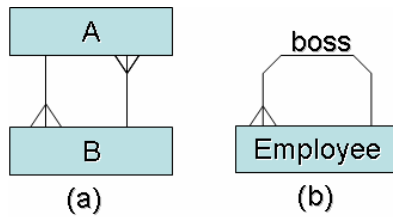


Figure 12 Recursive loops

In fact it is a mathematical theorem (the 'covering forest theorem' in graph theory) that this can always be done. Unfortunately this theorem is only true if the model does not contain *'recursive loops'*. A recursive loop is a loop where one can walk around following the direction of the relationships. If we look at Figure 12(a) we have a loop in which we can cycle forever ($A \rightarrow B \rightarrow A \rightarrow B \rightarrow \dots$). Note that Figure 10 does contain a loop ('Journey' \rightarrow 'Country' \rightarrow 'City' \leftarrow 'Journey') but this one is not recursive. You do *not* cycle forever *according to the direction of the relationships* involved. It is an *innocent* loop in the sense that it can be resolved by aliasing. Recursive loops are also called *mean* loops, because they prevent the application of the mathematical theory mentioned before. On the other hand it turned out be not so bad after all. We can isolate these mean loops and take appropriate measures. Sometimes there is an additional business rule saying that we can cut off the endless going in the loop, so that we again can handle the situation with a finite number of aliases. Another possibility is suggested by the example of Figure 12(b). Here we have a self reference. The possibly exists not resolve this recursive loop, but create a dimension table with a self reference as well. If this self reference presents a (unbalanced) hierarchy then some end-user (OLAP-) tools allow you to construct a so-called parent-child hierarchy, which is fine.

Now back to the question whether we can generate 'suitable views' based on our business model. We already did the first step: unfolding the model. The next observation is that when we have a snowflake structure like the left part of Figure 11 we can 'fully denormalize' this structure to only have one very broad table. Or to put it another way: we can create a view on this structure that looks like one table. Such a view is called a 'One Attribute Set Interface' (OASI). To clarify this even more we give the OASI of our business model starting from the root entity type 'Journey' in the following table. We indicate the origin of each attribute.

Attribute Name	Origin of Attribute
DAY	Journey
FROM_CITYCODE	F-City
FROM_CITYNAME	F-City
FROM_COUNTRYCODE	F-Country
FROM_COUNTRYNAME	F-Country
FROM_REGIONCODE	F-Region
FROM_REGIONNAME	F-Region
FROM_THIRDWORLDDYESORNO	F-Region
FROM_POLITY	F-Country
FROM_CITYSIZE	F-City
TO_COUNTRYCODE	T-Country
TO_COUNTRYNAME	T-Country
TO_REGIONCODE	T-Region
TO_REGIONNAME	T-Region
TO_THIRDWORLDDYESORNO	T-Region
TO_POLITY	T-Country
NROFPAX	Journey

Indeed this list contains the attributes (in **bold**) needed in the SQL statement on page 10. Just for the sake of completeness we also give the OASI of the left snowflake of Figure 11 (based on the root entity type 'Stationing'):

Attribute Name	Origin of Attribute
BUSTYPECODE	BusType
CAPACITY	BusType
BRAND	BusType
CITYCODE	St-City
CITYNAME	St-City
COUNTRYCODE	St-Country
COUNTRYNAME	St-Country
REGIONCODE	St-Region
REGIONNAME	St-Region
THIRDWORLDYESORNO	St-Region
POLITY	St-Country
CITYSIZE	St-City
NUMBERSTATIONED	Stationing

By the way: these two sets of attributes are generated by use of the BIReady™ tool!
Both these One Attribute Sets Interfaces can be supported by appropriate star schemas.

3.4 Staging format

Besides the user views on the information we can also generate staging tables. Figure 13 illustrates the idea of a generic staging format. Suppose we have an entity type 'Employee' in our business model (business: social security). And this entity type has the following attribute types: Birth Date, Gender, Residential Status, Domicile and Branche. Furthermore an employee is uniquely identified by a so called 'Identification Code'. Now the tool can generate a staging table for data of this entity as follows: a Begin Date, End Date and Change Date Time is added and a column called 'source' is also added. Each row that is inserted in this table is supposed to contain messages with information about an employee. This can be information about all the attributes (for instance if it concerns a whole new employee), but it may be about one or some attributes as well. The latter will occur if the source is a kind of log file of the operational system. It is even possible that only the column 'Identification Code' is populated, in which case the message is merely "There exists an employee with Identification Code abc."

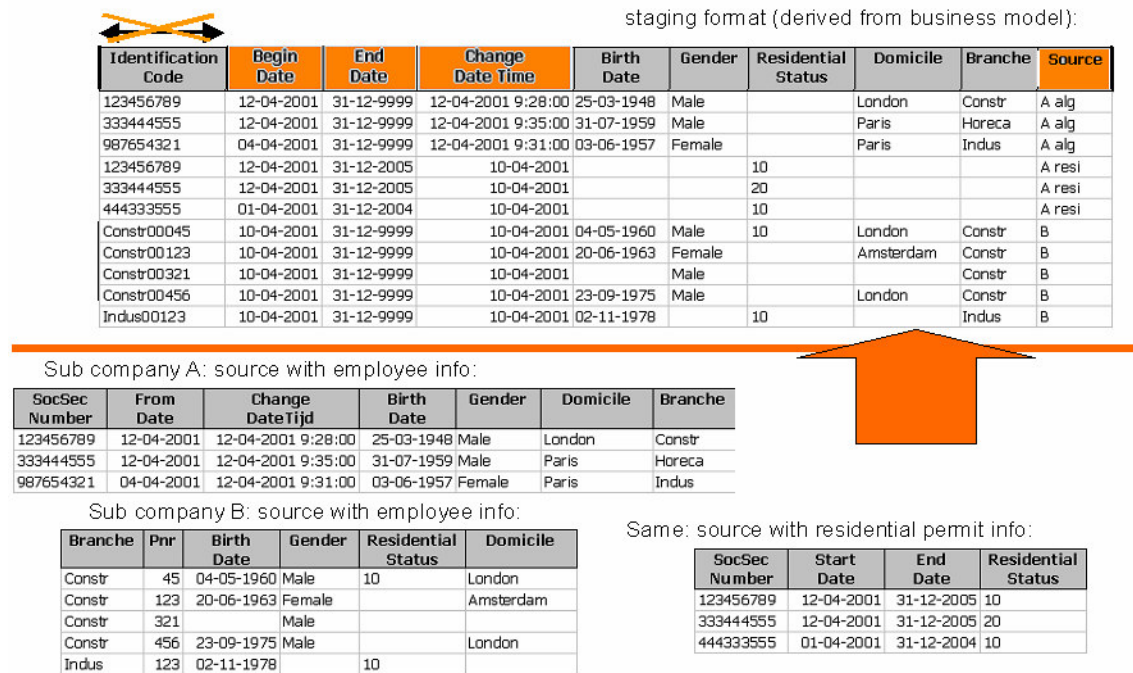


Figure 13 A generic staging table and how it can be populated from different sources

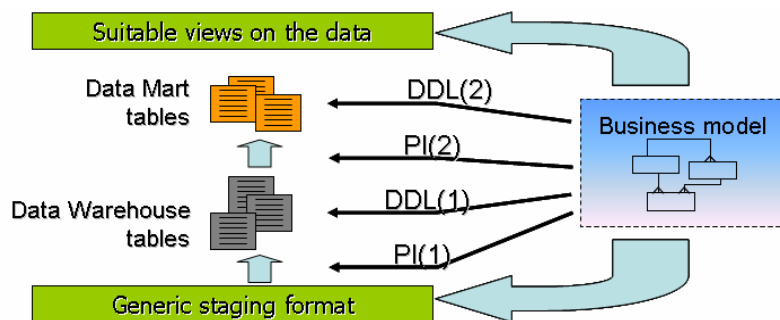
Now it must be possible to 'map' data from various sources to this staging structure. The main source system for employee information in this company has populated the first three rows. Note that the 'Residential Status' is missing. This was not foreseen at the time of design of this system. Meanwhile it has become a politically important issue (that is why it is in the business model), so an ad hoc system contains this information ("John's file"). The next three rows in the staging table come from this system. Note that here we have an explicit End Date. The message in the source is here: "The employee with social security Number 123456789 has a residential status of 10 as from 12/04/2001 to 31/12/2005". Note that adding this information to the staging table, we get duplicate Identification Codes. This is not a problem since this is handled by the process instructions in the black box afterwards. So, despite the fact that according to

the business model a person is identified by the 'identification code' we don't have a uniqueness constraint in the column 'identification code' in the staging table.

Now the organization is merged with another company. This sub company B has an operational system taking care of the registration of employee information of his own. We can get a kind of log file from this system and we populate our staging table from this source as well. What you clearly see here is the mechanism how data from different and heterogeneous source systems can be integrated: first they are mapped onto the same business staging format and secondly the real integration (taking care that all the information about one object is stored in the Data Warehouse in a consistent way) is effected by generic software.

3.5 Inside the transparent black box

In paragraph 3.1 we argued that two layers can be generated: (1) Suitable Views for end user applications and (BI-) tools, and (2) A business Staging Area and we described these layers in 3.3 and 3.4. Because both layers are generated, based on the business model only, all the functionality that has to be implemented between these two layers can be generated as well and exactly this observation comprises the power of the Model Driven Approach to Data Warehousing.



Now with BIReady™ the black box is not really black. It is transparent: you can see what is happening there. We call it a black box, because you don't have to do anything to implement it! But just for convenience and trust let us explain what the architecture is inside.

Figure 14 Inside the back box

The Data Warehouse tables are generated (DDL(1) in Figure 14), with the design principles that (1) it should be *relatively easy to adjust* the already running Data Warehouse when the business model changes. Furthermore the design of the Data Warehouse tables is (2) *optimal for the process* to keep the data up to date by the feeding processes from the business staging. The design of the Data Warehouse Tables can be uniquely derived from the business model and the requirements on history for individual attributes and/or relationships. See also paragraph 5.1. In advanced mode the DBA can have some influence on this design. For instance if one knows that an entity type has some frequently changing attributes, while other attributes are relatively stable. Then one can make sure that these groups of attributes are in different tables. With a particular choice of these parameters it is even possible to let BIReady™ generate a perfect Data Vault design (see Dan Linstedt: <http://www.tdan.com/i021hy01.htm>).

The trade off of the two design principles is, as is well known, that it is not that easy to query these DWH tables. Indeed they are not meant to be queried. For querying we have the Data marts. So we adhere to the CIF (Common Information Factory) principles of Inmomm and Imhoff.

The Data Mart Tables are generated (DDL(2) in Figure 14) in such a way that they are optimized for query use (generating cubes on top of them, installing reporting tools etc.). The design here follows the principles of dimensional modeling (star schema design) as promoted by Ralph Kimball. The design of these star schemas is highly supported by BIReady™. By using the Data mart Wizard (see paragraph 5.2) one only has to indicate some design decisions which can be depend on the RDBMS at hand.

And, last but not least, so-called process instructions (PI) are generated. These instructions (which are simply a set of derived Meta data also stored in tables, the so-called PI-tables) can be executed by a special 'engine': a piece of software designed to run on a variety of server platforms and to execute the generated instructions as efficient as possible. The instructions consist of three parts:

0. Mapping from sources to the Business Staging Tables, PI(0) in Figure 9;
 - a. Checking if data type conversion problems can arise, for example if a character field has to go into a date column;
 - b. If no data type conversion problems are expected and the source data are in the same environment, BIReady™ even creates a view on the source data instead of performing a time consuming mapping process;
1. Update the Data Warehouse Tables, PI(1) in Figure 14;
 - a. (internal) Change Data Capture;
 - b. Integration of data from different sources;
 - c. Creating and maintaining Surrogate Warehouse Keys;
 - d. Storing history;
 - e. Logging of the process;
2. Update the Data Marts, PI(2) in Figure 14.
 - a. Incremental update of dimension tables;
 - b. Denormalization, inheritance of history;
 - c. Creating and maintaining Surrogate Object Version Keys;
 - d. Loading of the fact tables;
 - e. Logging of the process.

On top of the star schemas Meta data can be generated for popular OLAP tools.

4 History of Data

In this chapter we pay attention to one of the most striking 'generic' issues: History of data. Most Data Warehouses keep track of history of data. And often this is done by a 'stacking' technique as described earlier. But sometimes the requirements are even more demanding. One wants to keep track of the history of history, the history of future, and so on. This is indeed a challenge for designers! But the good news is: it is generic! So it can be solved only once by a vendor.

4.1 History of administration

If we look at the first 3 rows of the table in Figure 16 it seems that we have a person, who first moved (on day 40 from Zip code 5000 to 7500) and then married (on day 50). But wait! If we look carefully we see that the column Start

Key	Start Date	End Date	Zip code	Civil Status	Marriage Date
1	20	40	5000	Unmarried	
1	40	50	7500	Unmarried	
1	50	9999	7500	Married	30
2	10	9999	7500	Married	10
3	10	9999	2400	Divorced	10

Figure 16 Stacked history

Date only gives the date on which the change is entered into the source system. The column 'Marriage Date' gives, at least for the attribute 'Civil Status', more information about the real start date of the change. If we assume that for the Zip code this real start date is about the same as the (recorded) administration date, we see that the real history of the person is the other way around: he (or she) first married and then moved! In Figure 15 we see the difference between the history of the administration and the real history of an object, at least that is what it now appears to be.

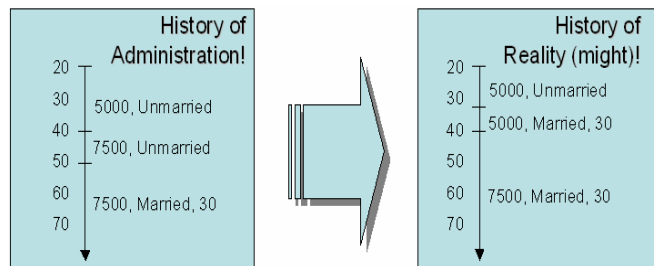


Figure 15 History of administration

4.2 History Types

Paragraph 4.1 showed that time issues can be precarious. Sometimes the source contains the real start date (time) of a change. Sometimes even the administrative change date time (a 'timestamp') is contained in the source. In these cases the possibility exists of changes with retroactive effect. Maybe one wants to keep track of both. For instance, the Sarbanes-Oxley Act could make this urgent.

For each attribute or relationship of an entity the business should be able to decide to have:

- No History (Known as Kimball Type 1);
- Simple History (Known as Kimball Type 2);
- History of History (*).

(*) is important in situations in which changes with retroactive effect may occur (Insurance companies for instance!). Example:

Questions asked in June 2005:

"Was customer Johnson married in February 2003 according to the data in our system in September 2004?"

Answer: **Yes**

"Was customer Johnson married in February 2003 according to the data in our system in January 2005?"

Answer: **No**

Note that Kimball perceives 3 types of history tracking in dimensions. His type 3 is pretty ad hoc and not considered here.

5 An Example: BIReady™

A concrete example is probably the best way to illustrate matters. In the following paragraphs you will see a number of screens of the BIReady™ product.

5.1 Business Model

In Figure 17 you see a part of the business model (in this case the business model was obtained by reverse engineering the Microsoft Northwind demo database). You are

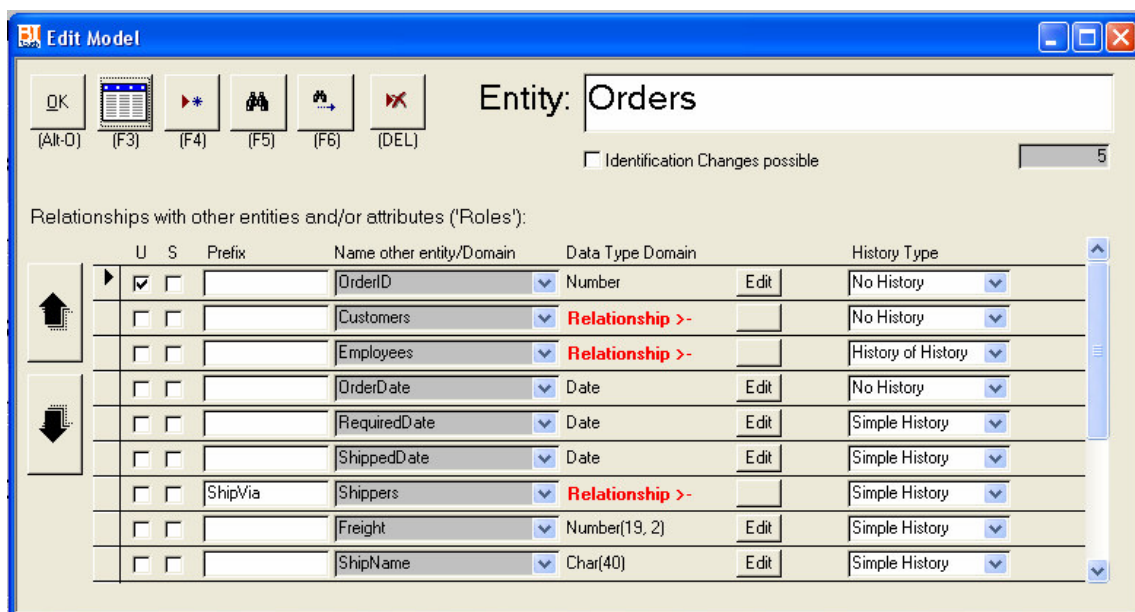


Figure 17 One Entity type with relationships and history requirements

looking at the entity type 'Orders' together with relationships to other entity types and attributes. For each attribute and/or relationship one can specify the type of history that has to be kept in the Data Warehouse. This is the only Meta Data that is required for the generation of the Enterprise (central) Data Warehouse!

In a more advanced mode some more design decisions can be indicated. For instance if one knows that some attributes are frequently changing, while others are only slowly changing. In that case it makes sense to store them physically in different tables.

5.2 Data Mart Wizard

For the design of the Data Marts some additional design decisions have to be made. A wizard can support this (design-) process. First of all the (potential sources for) the fact

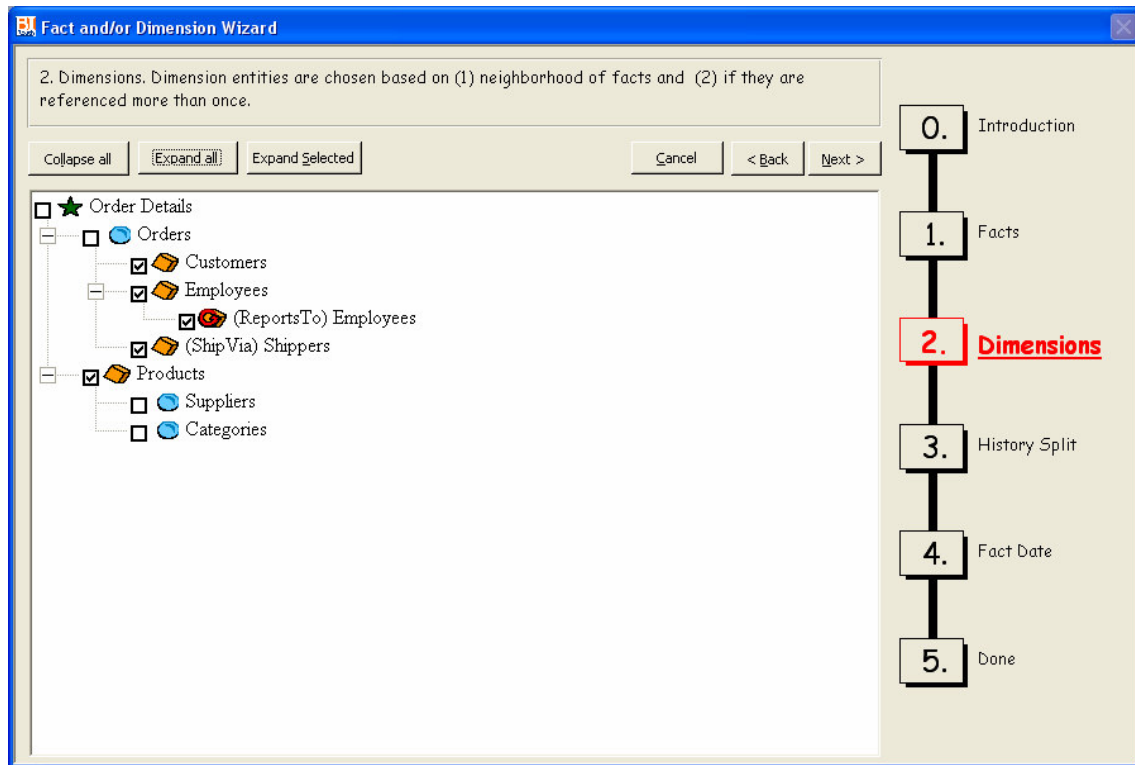


Figure 18 The Datamart wizard

tables have to be discovered (step 1). Secondly (shown in Figure 18) we have to establish the (roots for the) dimensions. The first proposal of the tool in this example is that there should be an 'Orders' dimension and a 'Products' dimension (and the rest should be denormalized into these two). With the latter (the 'Products' dimension) we agree, but the 'Orders' dimension seems to be too big a dimension. So we uncheck the Orders in this screen. Then the tool guesses that we probably want to have Customers, Employees and Shippers as dimensions. We judge this as an excellent idea, so we click 'Next'. The remaining steps are less illustrative.

Note that, leaving them unchecked, means that the entity types 'Suppliers' and 'Categories' will be denormalized into the 'Products' dimension.

5.3 Dimension and Facts Edit

After you used the Data Mart wizard you can edit the Meta data of the Data Marts even further. You can add remove and edit dimensions and/or fact tables. The transaction date

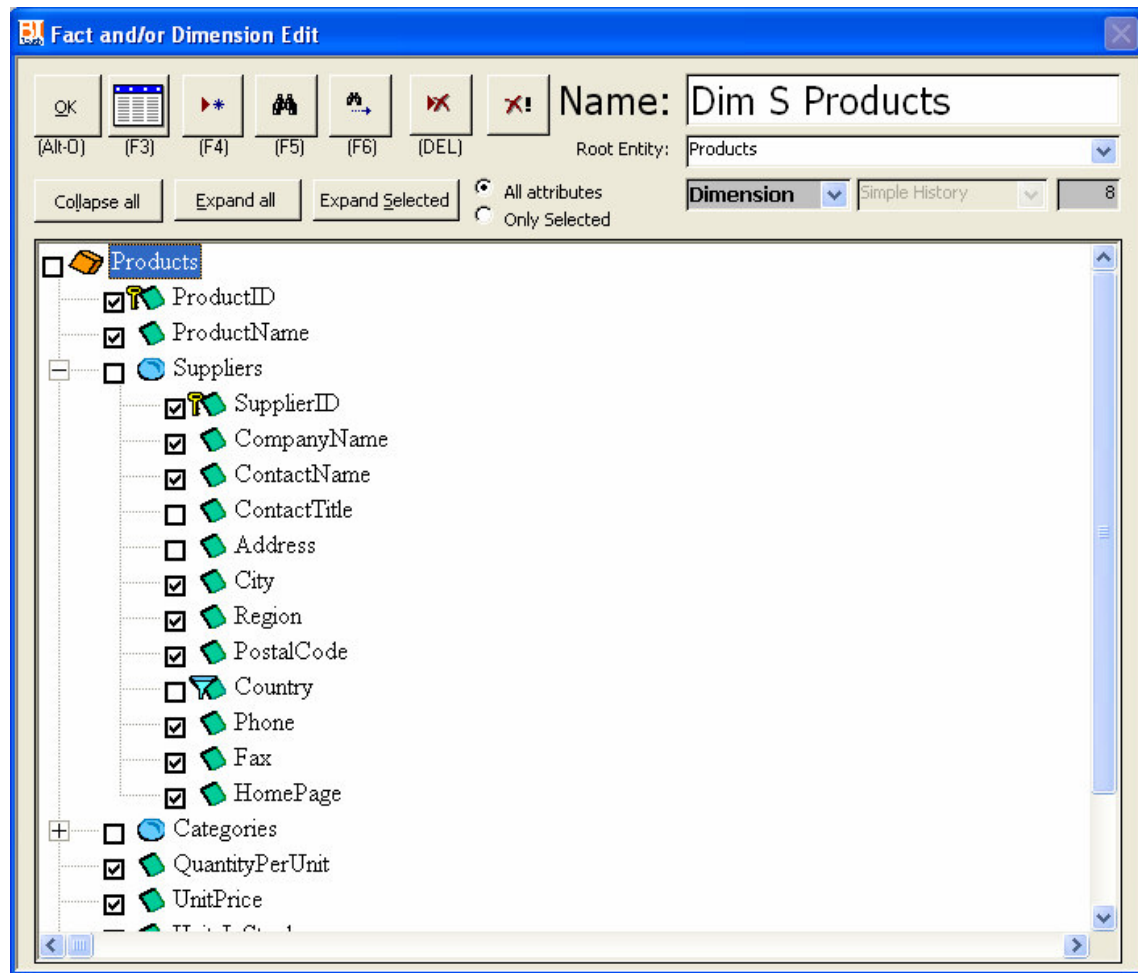


Figure 19 Editing the Meta data of a dimension

for the facts can be altered and so on. In Figure 19 you can see that there is apparently no need for 'ContactTitle', 'Address' nor 'Country'. By right clicking on an attribute one can define a filter condition. This in case one wants to have a Data Mart with restricted data in it. In the example we are only interested in sales of product which are supplied by suppliers from the UK. This filter condition can be specified with the attribute 'Country'. Attributes with a filter can be identified by the funnel symbol.

5.4 Generated Star schema

In Figure 20 the (structure of the) Data Mart that is generated for the 'Northwind' case is shown. It is generated in Microsoft SQL Server and shown via the diagram wizard of Microsoft SQL Server. Note that each dimension has a VERSION_ID as primary key. This



Figure 20 The resulting star schema

is Data Warehouse Key (meaningless number) identifying one *version of* an object. This key (pointer!) is also the foreign key in the fact table (e.g. S_CUSTOMERS_VID). But both in the fact table and in the dimension tables we also have the OBJECT_ID. This is the Data Warehouse Key identifying the object itself. Normally one will join the fact table and a dimension table by the version id, because most of the time one wants the transactions together with the dimension object info as it was at the time of the transaction. But the object id's (which are already present in the Data Warehouse) allow ad hoc querying.

5.5 Mapping Screen

To support the mapping from real operational sources to the staging area, simple mappings can be defined and maintained. So ETL is supported and often this will suffice.

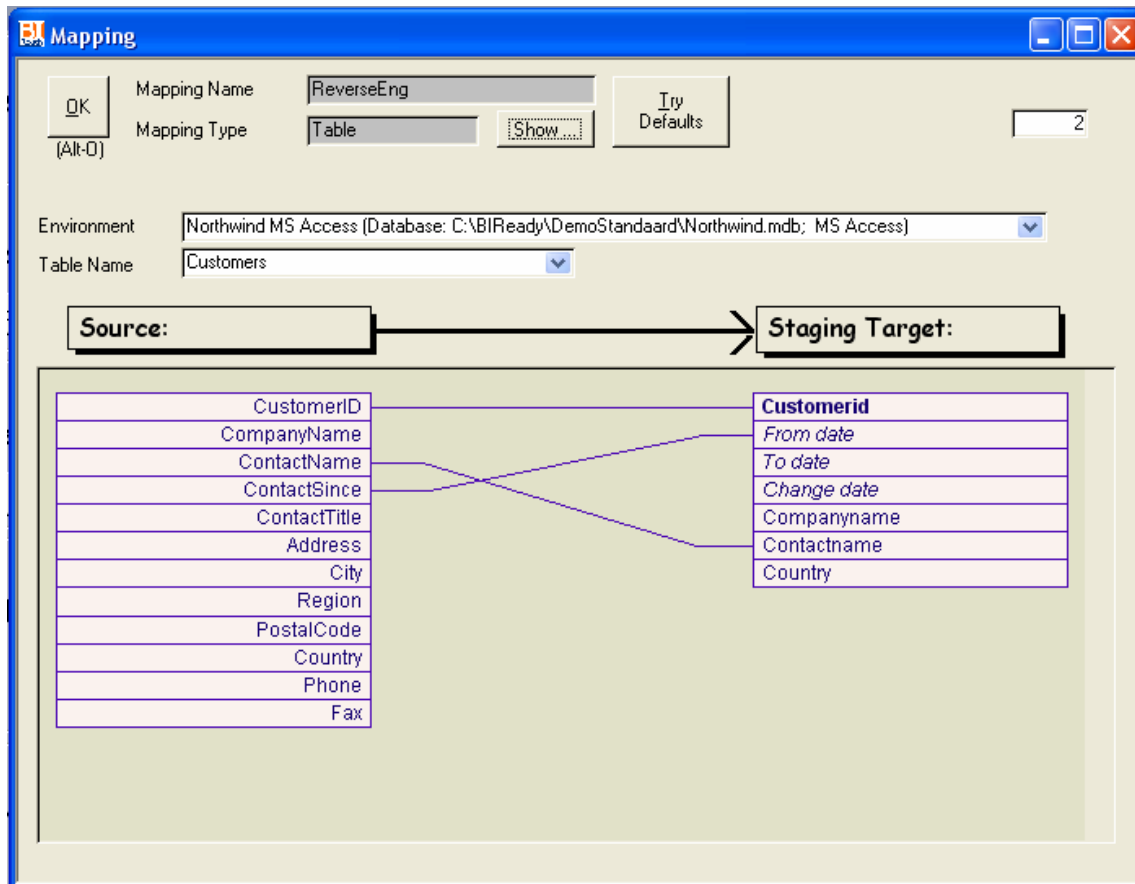


Figure 21 Screen for one mapping

In some situations one can use the ETL functionality that comes with the database products if desired (for example OWB for Oracle or MSIS for SQL Server) to populate the business staging tables.

The example of Figure 21 shows the case where we need one separate mapping only for the attribute 'ContactName'. This is so, because in the source data an explicit date field 'ContactSince' exists, which tells us as from which date the person is in charge as our contact within the customer. We want to use this date as effective date and we can accomplish this by drawing a line from 'ContactSince' on the left hand side to the staging field 'From date'.

5.6 Process Instructions

The start of the generation of the process instructions is accompanied by a window shown in Figure 22. The process instructions are stored in tables. These tables are read

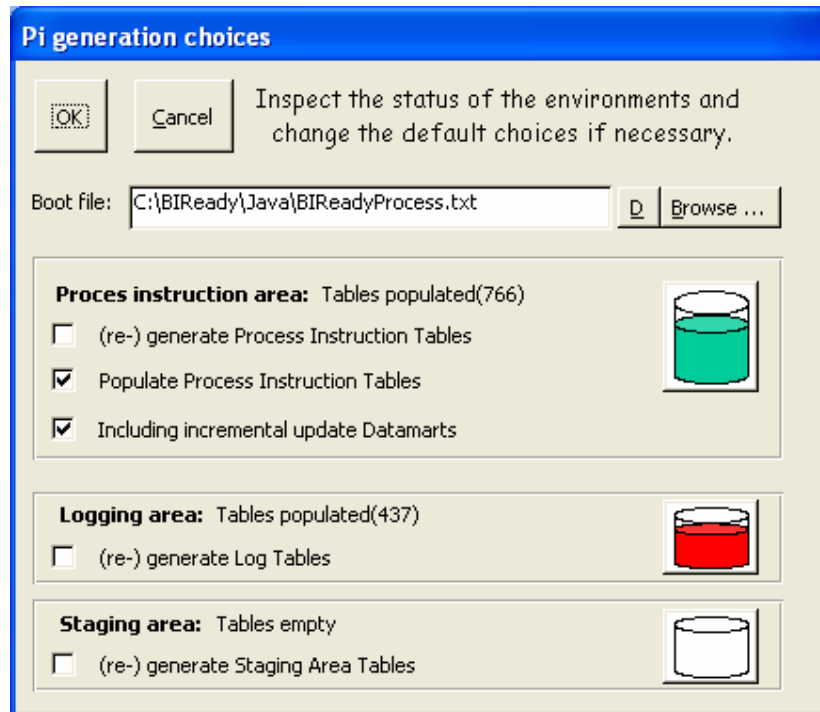


Figure 22 Start of process instruction generation.

and executed by the Process Engine. In the same screen you can indicate if you want to (re-) generate staging and/or logging tables. The barrel signs indicate that there are already tables present. That is why by default the '(re-) generate' checkboxes are unchecked. If they are white the tables are empty. If the barrel seems to be filled, then there are rows in the tables. If the color is green, it is no problem to drop the tables, because they contain derived data (like process instructions) which can be easily computed again. If the color is red, then one might lose information dropping the tables.

5.7 Starting the process

You can start a process online or just generate a command file, such that the processes can be scheduled overnight. Either way you can specify some parameters. This is shown in Figure 23.

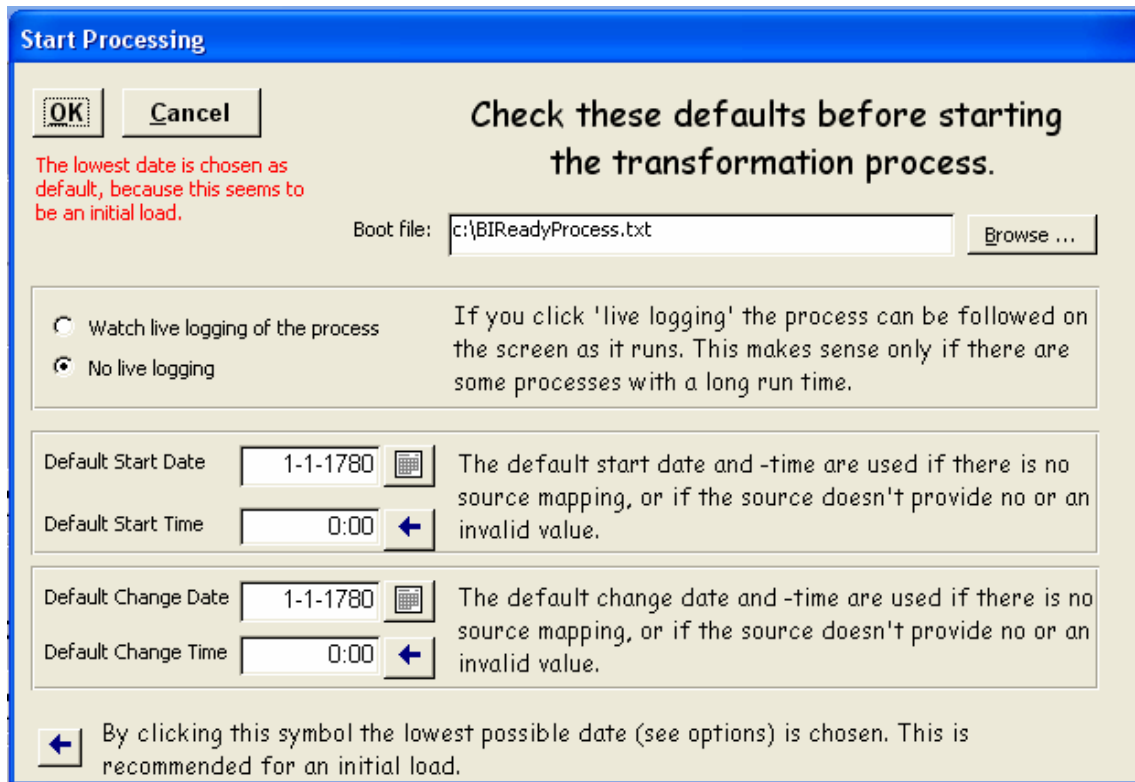
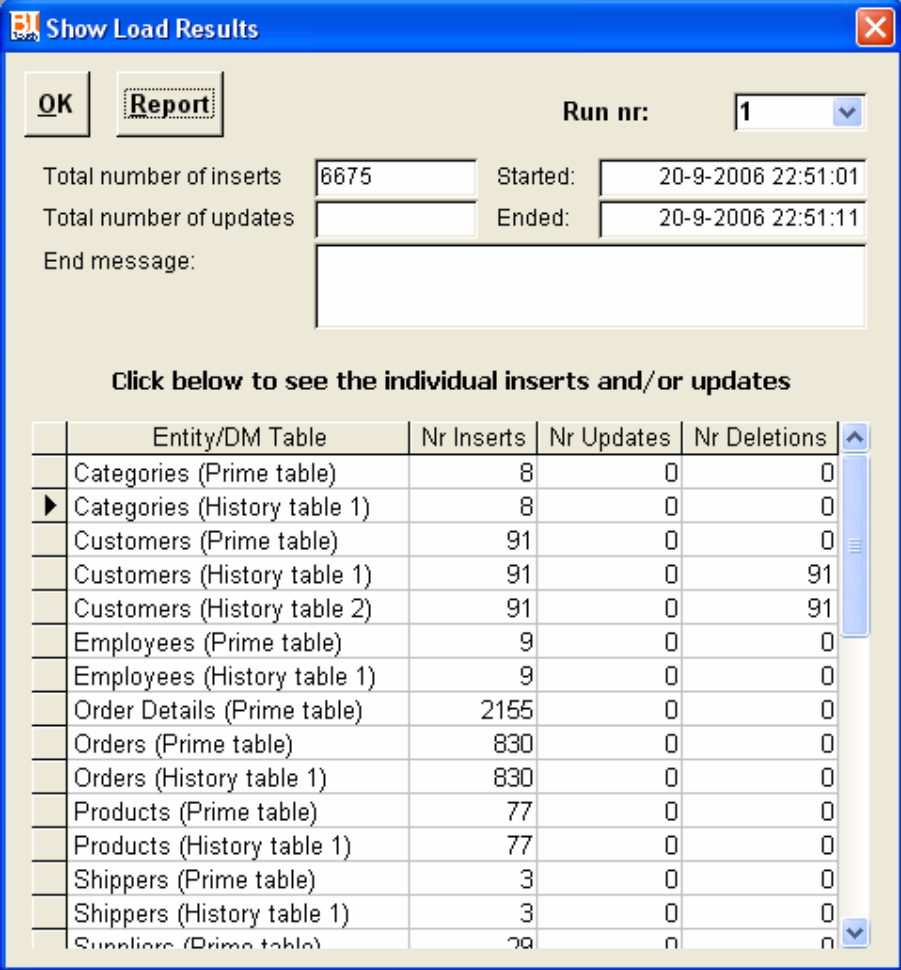


Figure 23 Parameters for a particular run of the engine

5.8 Log Report of Run

After each run one gets a report of what has happened in that run. One can drill down in this screen to see more details for instance the individual inserts or some examples of



Show Load Results

OK Report Run nr: 1

Total number of inserts: 6675 Started: 20-9-2006 22:51:01
 Total number of updates: Ended: 20-9-2006 22:51:11
 End message:

Click below to see the individual inserts and/or updates

Entity/DM Table	Nr Inserts	Nr Updates	Nr Deletions
Categories (Prime table)	8	0	0
▶ Categories (History table 1)	8	0	0
Customers (Prime table)	91	0	0
Customers (History table 1)	91	0	91
Customers (History table 2)	91	0	91
Employees (Prime table)	9	0	0
Employees (History table 1)	9	0	0
Order Details (Prime table)	2155	0	0
Orders (Prime table)	830	0	0
Orders (History table 1)	830	0	0
Products (Prime table)	77	0	0
Products (History table 1)	77	0	0
Shippers (Prime table)	3	0	0
Shippers (History table 1)	3	0	0
Suppliers (Prime table)	3	0	0

Figure 24 Logging information about one run.

them) This example screen (Figure 24) shows the initial load for the Northwind Case.

6 Conclusions

We have explained how the process of designing and building a Data Warehouse and Data Marts can be improved. This is based on the observation that a lot of issues is generic by nature. We have discussed these in detail and they are listed in Figure 25.

Of course not all the issues in Data Warehousing or Business Intelligence are Generic. That is why we do not claim 100% savings by using a Model Driven product. But savings of 50% of time and costs are observed.

Furthermore the savings are even higher in the maintenance phase. Because then it turns out that only the business model has to be changed, according to the new requirements.

Key issues in Data Warehousing

... and which of them have a generic nature









	1. History	Including 'History of History'
	2. Data Integration	Heterogeneous data sources.
	3. Performance	Load Times, Response Times and Data Compression.
	4. Flexibility	The ability to change an existing DWH; real 'start small, think big'
	5. Logging	To log all the loading activities
	6. Generating the DWH itself!	

Figure 25 Key issues in Data Warehousing

BIReady is the only product that offers a complete Model Driven Approach to Data Warehousing and at the same time adheres to best established standard architectures like Common Information Factory (CIF) and Star Join Schema design.