Slowly Changing Measures

Mathias Goller
Johannes Kepler University
Altenberger Str. 66b, Linz, Austria
goller@dke.uni-linz.ac.at

Stefan Berger
Johannes Kepler University
Altenberger Str. 66b, Linz, Austria
berger@dke.uni-linz.ac.at

ABSTRACT

In data warehousing, measures such as net sales, customer reliability scores, churn likelihood, or sentiment indices are transactional data scored from the business events by measurement functions. Dimensions model subject-oriented data used as analysis perspectives when interpreting the measures. While measures and measurement functions are traditionally regarded as stable within the Data Warehouse (DW) schema, the well-known design concept of slowly changing dimensions (SCDs) supports evolving dimension data. SCDs preserve a history of evolving dimension instances, and thus allow tracing and reconstructing the correct dimensional context of all measures in the cube over time.

Measures are also subject to change if DW designers (i) update the underlying measurement function as a whole, or (ii) fine-tune the function parameters. In both scenarios, the changes must be obvious to the business analysts. Otherwise the changed semantics leads to incomparable measure values, and thus unsound and worthless analysis results.

To handle measure evolution properly, this paper proposes *Slowly Changing Measures* (SCMs) as an additional DW design concept that prevents incomparable measures. Its core idea is to avoid excessive schema updates despite regular changes to measure semantics by a precautious design, handling the changes mostly at the instance level. The paper introduces four SCM types, each with different strengths regarding various practical requirements, including an optional historical track of measure definitions to enable cross-version queries. The approach considers stable business events under normal loading delays of measurements, and the standard temporality model based on the inherent occurrence time of facts. Furthermore, the SCMs concept universally applies to both, flow and stock measure semantics.

Categories and Subject Descriptors

H.2.1 [**Database Management**]: Logical Design—*Data models*; H.2.7 [**Database Management**]: Database Administration—*Data warehouse and repository*

Keywords

Data Warehouse Modeling and Design; Evolving Measurement Functions; OLAP Modeling

1. INTRODUCTION

In data warehousing, the database is conceptually divided into facts and dimensions [8, 9]. Fact data corresponds to summarized quantitative measurements of *business events* and their properties. Dimension data, in contrast, are qualitative attributes stored along with the measures in the Data Warehouse (DW). Dimensions represent the business entities or *subjects* which altogether specify the semantic context of the measures, enabling a meaningful interpretation of facts.

The instances of dimensions—called *members*—compose the multi-dimensional analysis space needed to aggregate the measures of the (hyper-)cube data inside the DW. *On-Line Analytical Processing* (OLAP) tools provide the business analysts with typical operations such as roll-up to compute key performance ratios, and create reports. At physical level, the *facts* of an n-dimensional DW schema are tuples comprising n members—one per dimension—plus m measures. Each n-tuple identifies a unique cell of the hypercube; the measures are stored as numerical values in the cells, and associated with a value domain (e.g., the Euro currency).

Slowly changing dimensions (SCDs) is a well-known concept in dimensional modeling of DWs that accounts for the *potential volatility* of dimension members. In order to keep the members up-to-date, SCDs manage updates to their instances, and optionally track the update history according to the designer's choice. SCDs thus help distinguish true changes of member features from simple corrections of erroneous data. Kimball [9] described three increasingly powerful SCD types, as explained below in Subsection 1.2. In many practical settings the subjects modeled by dimensions evolve and change unpredictably over time. For instance, consider the typical example of customers of a retail company. It is quite common that persons change their address or phone number, whereas other personal attributes such as marriage status are much more stable, and some will never change such as the birth date.

Measurement functions compute the measure values of facts during the Extract, Transform, and Load (ETL) process from transactional data on business events by scoring their commercially interesting properties. Often the measurement functions are complex, highly specific to the DW model, maybe even secret, and influenced by the organization that runs the DW, its business sector, its commercial goals, or the analysts' individual preferences. For instance, customer satisfaction score, customer churn likelihood, or the personal sentiment about some given brand are calculated by parameterized scoring algorithms.

The traditional simplifying assumption that measures in the DW schema remain stable is commonly disproved by practical requirements. On the one hand, the business events might change after their occurrence in some application scenarios, denoted *late registration* [6] as explained in Section 4. On the other hand, the measurement functions might change over time, and hence the summarized scores computed from the underlying events. In practice these changes often remain unrecognized as current DW models and tools lack appropriate support for changing measure definitions. In this paper we assume stable business events, and instead consider the latter case of changing measure semantics.

Complex measures are the prime candidates for changes over time since their computation is influenced by numerous factors. For instance, the *sentiment* of an individual towards a specific topic is generally scored from the documents authored by this person (e.g., postings in social networks). Basic sentiment analysis algorithms simply count the number of positive and negative opinion words, and classify the sentiment accordingly, but much more sophisticated algorithms are constantly published.

Even simple measures like the quantity of units sold per day, product, and branch are subject to change in practical scenarios. The quantity sold might include or exclude canceled sales transactions, or sales on consignment. We observed several changes in the measurement function of *net sales in local currency* in a large Austrian retail chain within five years, caused by the introduction of the Euro currency, a rising VAT, a newly imposed recycling tax, and several customer bonus programs. Section 2 will illustrate the possible effects on the net sales measure induced by such changes.

There are two ways of evolving the measure definitions in ETL processes: either re-define the whole measurement function, or fine-tune its configuration parameters. In both cases, the measures computed in ETL cycles subsequent to the function update will be semantically different from the pre-existing measures, so that any change in the measure definition should be treated as an update of the conceptual DW schema. Without proper documentation, the analysts could not even notice the changed semantics, and that the measures have actually become incomparable—like monetary amounts of unknown currency. The OLAP analysis becomes unsound, yielding worthless results that might cause false conclusions and poor business decisions.

In order to avoid the potential problem of incomparable measures, we propose to carefully document all changes to the measurement functions with *Slowly Changing Measures* (SCMs) as an additional DW modeling concept. While regarding all measure definition updates as an evolution of the DW schema, our approach handles these updates mostly at the instance level. By choosing the most appropriate options according to the analysis requirements at design time, SCMs help DW designers avoid excessive schema updates when the changes occur, as sketched in the next subsection.

1.1 Scope and Outline

Slowly Changing Measures (SCMs) manage updates to the measure definitions in DW schemas while ensuring consistent measure semantics. Our approach assumes stable business events, normal loading delays in ETL, and simple temporality based on the occurrence time of facts that is inherent to DW models given their historical nature. In contrast to the late registration problem [6], the summation class of measures—*flow* or *stock*—is secondary for SCMs, as the changes we consider apply to a more abstract modeling layer. Specifically, late registration DWs must correctly summarize the measures from volatile events after some delay, whereas SCMs must correctly handle the changed definitions of summarized (flow or stock) measures. Hence, the SCMs concept applies universally to measures of both summation classes.

The key idea of our approach is the precautious design of the exact semantics of potential changes to measures in the DW schema. We introduce four types of SCMs, each specifying a strategy and set of options for the change history management of measure definitions with minimal impact at the physical DW level. These SCM types address the most common practical requirements. As soon as the changes appear they can be handled mostly at the instance level according to the strategy chosen beforehand.

This paper presents the following contributions:

- SCMs support updates to measurement functions over non-changing business events with minimum design effort. Each update is regarded as an evolution of the cube schema, but handled mostly at the instance level.
- SCM Type 3 inherently supports *multi-version* queries—i.e., across several historical definitions of the same measure—without a bi-temporal model.
- Hence, SCMs are simpler than Temporal or Multi-Version DWs, yet sufficient for most practical cases. The SCM approach can be used in standard DW models, and implemented in existing ROLAP platforms.

The remainder of this paper is organized as follows: first we revisit Kimball's SCDs in the following subsection. Section 2 introduces potential applications of slowly changing measures which serve as running example throughout the paper. Section 3 describes the approach of slowly changing measures in detail, introducing four SCM types, illustrating them all with the running example, and discussing their pros and cons. Section 4 discusses related work, and finally Section 5 concludes the paper.

1.2 Slowly Changing Dimensions Revisited

Ralph Kimball, one of the most notable contributors to practical DW solutions, introduced three types of slowly changing dimensions (SCDs) [9]. The general purpose of SCDs is to preserve the referential integrity between facts and dimensions in hypercubes while supporting changes in dimension members. The Type 2 and Type 3 SCDs also track historical states of members and their attributes.

SCD Type 1: Overwrite. Type 1 of SCDs equals the traditional do-nothing approach, and thus is the simplest solution to managing dimension changes. SCDs of Type 1 simply overwrite the attributes of members upon updates without keeping any history. Clearly, this approach should only be used if the previous states of members are irrelevant—e.g., updating the birthdate of a person is assumed to correct an erroneous entry, whereby the previous value is useless.

SCD Type 2: Additional member tuple. Type 2 of SCDs create an additional tuple for members on every update. It is necessary to complement the dimension's primary key with a surrogate key. Intuitively, the surrogate key gives the version number of the member record. To maintain referential integrity, facts refer only to the dimension's surrogate key. Hence, the DW maintains the historically correct dimensional context for all facts in the hypercube. The price of maintaining the full member history in Type 2 SCDs is a considerable overall growth in dimension size.

SCD Type 3: Additional old column. Type 3 of SCDs introduces an additional column for every "updateable" member attribute. For each such attribute the Type 3 SCD provides both, the current and original version. Another variant of Type 3 SCD records the current and *previous* version of the member attribute [7], but the key idea is the same in either case: use two columns for one attribute. While this approach avoids increasing the number of tuples, it modifies the dimension schema by doubling the attributes whose original state is important. In practice this approach is rarely used, though, because its usefulness is limited in case of frequent member changes [13].

2. MOTIVATING EXAMPLE SCENARIO

To illustrate the problems arising from changing measure definitions we assume a fictitious retail company analyzing sentiment data and point-of-sales (POS) data in a DW. Let us consider a simple data mart comprising two measures, *sentiment* and *net sales*, the brand dimension of products, and the time dimension. The ETL process records net sales per brand from the POS cash registers, as Figure 1 shows. Furthermore, the company monitors social media to analyze the reputation of each brand in its retail assortment. Its ETL process follows new postings that mention one of the brands, and scores their messages using a simple sentiment measurement function, illustrated in Figure 2.

At some day W the retail company introduces the following marketing scheme: every purchase over 100,- earns the customers a bonus voucher worth 10,- for their next purchase. The marketing scheme successfully encourages the customers to buy more often. Internally, the cash register books the voucher as a means of payment, and allocates the discount as costs to the product(s) bought in the original transaction. Consequently, the net prices of the products remain constant; the net sales measure increases according to the new customer behavior, as charted in Figure 3.

Later, on day X the company decides to change the internal booking of the discount. From that day on it regards the bonus voucher as a price reduction. That calculation scheme reduces the net price, changing the net sales semantics, and hence the measure decreases accordingly. As Figure 4 illustrates, the net sales measure drops even though we presume the same sales transactions for both days. Figure 3 visualizes the sudden gap in the net sales curve caused by the changed booking of the discount. While for the customers everything remains the same, note that the sales measure appears to signal a changed customer behavior after day X, but clearly this would be a misinterpretation of reality.

In order to score the sentiment measure, the retail company employs the most basic sentiment analysis algorithm. It simply counts the occurrences of positive and negative opinion words from two given input word lists, and subtracts the negative count from the positive count to obtain the final measure value (0 indicates a neutral or undetermined sentiment). Figure 2 sketches the sample data mart on day X: a scheduled query has retrieved the new messages mentioning brandA, as shown at the top. Using the input word lists depicted in the middle, the sentiment measurement function obtains the message scores given at the bottom.

On day X the retail company also tunes the sentiment measurement function. In our case this is easy to achieve by adding or removing sentiment words from the input word lists. Assume that the DW designer adds the the word *sick* to the list of negative opinion words, and on day X+1, the ETL process retrieves a new message on brandA. Although both messages contain the same opinion word sick, it is only considered for the sentiment score of message ID 2 due to the changed measurement definition, as shown in Figure 5.

In our scenario the following problems occur at day X+1: (i) The net sales measure has suddenly dropped, and also the sentiment measure for brandA has decreased. From the above explanations it is clear that this happened as mere coincidence. Without a proper documentation of the measure semantics, though, the business analysts could falsely conclude that net sales have decreased due to a more negative customer sentiment towards brandA. (ii) The sentiments scored for messages with ID 1 and 2 have become incomparable due to the measurement function tuning. In the data mart, however, it should become obvious that both these messages contain the same opinion words.

In the later sections we will discuss how the different types of SCMs can overcome these problems, and revisit the sample scenario to illustrate our design solutions.

Cash register data of day X

Brand	Date	Gross sales	Tax	Discount
BrandA	X	120,-	20,-	
BrandA	X	24,-	4,-	10,-

⇓

Measurement Function: Net sales := Gross sales – Tax

⇓

Data mart after ETL cycle of day X

Brand	Date	Net sales	Sentiment
BrandA	X	120,-	0

Figure 1: Original net sales measurement function

Query result (social media staging table)

Search string	ID	Date	Retrieved message
BrandA	1	X	I am sick of the new ad of BrandA.

⇓

Parameters of measurement function (opinion word lists)

Positive	cool, good, great, well
Negative	awful, bad, forget, ill

⇓

Message scores

Search string	ID	Date	Sentiment
BrandA	1	X	0

Figure 2: Original sentiment measurement function

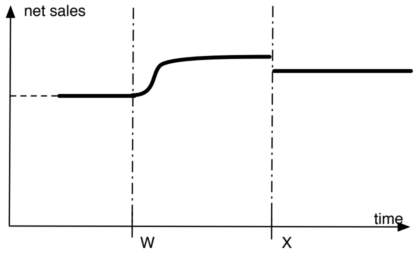

Figure 3: Net sales – measure curve

Cash register data of day $X+1$

Brand	Date	Gross sales	Tax	Discount
BrandA	$X+1$	120,-	20,-	
BrandA	$X+1$	24,-	4,-	10,-

⇓

Measurement Function: Net sales := Gross sales – Tax – *Discount*

⇓

Data mart after ETL cycle of day $X+1$

Brand	Date	Net sales	Sentiment
BrandA	X	120,-	0
BrandA	$X+1$	110,-	-1

Figure 4: Changed net sales measurement function

Query result (social media staging table)

Search string	ID	Date	Retrieved message
BrandA	1	X	I am sick of the new ad of BrandA.
BrandA	2	X+1	BrandA is making me sick.

⇓

Parameters of measurement function (opinion word lists)

Positive	cool, good, great, well
Negative	awful, bad, forget, ill, *sick*

⇓

Message scores

Search string	ID	Date	Sentiment
BrandA	1	X	0
BrandA	2	$X+1$	-1

Figure 5: Changed sentiment measurement function

3. MANAGING CHANGE IN MEASURES – SLOWLY CHANGING MEASURES

Nowadays the constant evolution of typical data warehousing application domains, along with rapid advances in DW technology and data analysis, are rising the requirements towards a more flexible notion of measures. DW designers commonly adapt measurement functions to fine-tune the ETL process over time, but this requires appropriate design techniques. As explained in Section 2, changing measure definitions compromise the soundness of OLAP analysis, and thus cannot be ignored. Only under the specific circumstances explained below in Subsection 3.1, *consciously* hiding changes in measure definitions may be useful. Nevertheless, the do-nothing approach still dominates in practice, and is responsible for the problems discussed in this paper.

In order to allow sound and insightful OLAP analysis, the semantics of even the simplest measures must be modeled properly. The four SCM types proposed below pre-define various design options for managing the evolution of measure definitions in existing DW models with minimum changes. Types 0 and 1 are the simplest possible solutions that leave the original cube schema untouched. While Type 0 allows untracked changes under exceptional circumstances, Type 1 ensures consistent measures by re-computing outdated values. These two types are appropriate whenever the history of measure definitions is insignificant. In contrast, Types 2 and 3 document the historical track of measure definitions. Type 2 introduces a version dimension, whereas Type 3 amends a separate new measure attribute to the original cube schema. Hence, Type 3 is the only solution inducing schema updates when measure changes occur.

At design time, the proposed approach demands a careful analysis of *foreseeable changes*, and an appropriate choice of SCM Types. The particular SCM type allocated to each measure is a fundamental design choice, and like its underlying data type it is final. Altering the SCM type would entail re-initializing the entire hypercube. It is possible, though, to combine various SCM Types in the same cube schema.

Table 1 gives an overview of the SCM Types we propose, aligns our approach to slowly changing dimensions [9], and recommends some major design guidelines. Obviously our work is inspired by Kimball's, and so each SCM type follows the semantics of its analogous SCD counterpart whenever appropriate. The following subsections will explain each of the SCM Types 0 to 3 in more detail, and summarize the most important consequences of the different solutions.

3.1 Type 0: Conscious Do-Nothing

We denote the conscious do-nothing strategy as Type 0 of SCMs which, intuitively, allows untracked changes of a measure definition necessary to maintain its *original* semantics. Note that in dimensional modeling there is no explicit notion of "type 0 slowly changing dimension" because this simply corresponds to correcting erroneous data. Otherwise, hiding semantic changes of dimension members would even break the proper dimensional context of the measures, and hence compromise the soundness of OLAP analysis.

As the previous sections showed, traditional cubes commonly contain incomparable measures. The top left of Figure 6 depicts our sample data mart, regarding both measures as SCM of Type 0. There is no obvious hint or documentation allowing the analysts to recognize the semantic changes of the measures at day X. As the measures appear consistent, the analysts might falsely correlate the decreased net sales with the worse customer sentiment towards brandA. Typically, Type 0 SCMs cannot solve the problems mentioned in Section 2, and thus should strictly be avoided.

In rare cases, though, the SCM Type 0 can be employed to explicitly maintain the original measure semantics despite the occasional tuning of its underlying measurement function, such as in sentiment analysis with buzzwords. These *consciously untracked* measure re-definitions look dangerous at first sight, but prove useful for measuring volatile value domains. Human speech, for instance, is very dynamic—new words and terms arise and disappear, and it might be necessary to adapt the measurement functions accordingly.

We denote this phenomenon as the *buzzword problem* since it is particularly urgent in sentiment analysis. Often the DW designer might wish to add or remove opinion words as input for the sentiment analysis algorithm, but signal an unchanged measure semantics nevertheless. Consider the term *Deepwater Horizon* as an example opinion word. Before 2010, social media postings containing Deepwater Horizon were probably neutral, but later on the term likely indicated a negative sentiment towards the oil industry. For such well-justified cases, SCM Type 0 is an appropriate and elegant solution. Even so, in most cases reverting to one of the other SCM Types is safer and preferable.

3.2 Type 1: Re-Compute/Initial Load

SCMs of Type 1 always reflect the most recent measurement function, ignoring any previous definition. In order to maintain the facts comparable, any changed measure definition triggers the re-computation of all older values. The older outdated measure values are overwritten.

The complete re-computation of older measures practically resets the hypercube data, and hence we also denote Type 1 SCMs as the *initial load* strategy. Clearly, this approach cannot track the history of measures, and should

Type	SCM / *strategy*	SCD [9]	Common Features	Guidelines (Use When?)	Data Retention
Type 0	*Conscious Do-nothing* (overwrite nothing)	—	Unchanged semantics with a changed definition; otherwise: Corrupted facts (measures become incomparable)	Only use for well-justified "untracked" changes (e.g., buzzword problem)	Irrelevant (not needed)
Type 1	*Initial load* (Re-compute, over-write measures)	Overwrite old value	Previous state gets lost, only recoverable with retention of operational data	No history needed—e.g., new preferable scoring function; error corrections	Essential
Type 2	*Proactive versioning* (version flag/dimension)	Additional member tuple (surrogate key)	Previous state remains untouched, is fully trackable	Historical track needed, multi-version OLAP queries, frequent changes—e.g., complex/preliminary functions	Irrelevant (not needed)
Type 3	*Lazy amendment* (additional measure attribute)	Additional old column (for first or previous attribute)	Additional old measure (old measurements; new measurements = old attribute)	Both, original and current version needed—infrequent changes only	Necessary for retrospective measurements

Table 1: Slowly Changing Measures and Dimensions by Type – SCM design guidelines

Sentiment Type 0: conscious do-nothing

Message scores

Search string	ID	Sentiment
BrandA	1	**0**
BrandA	2	**-1**

Data Mart Type 0

Brand	Date	Net sales	Sentiment
BrandA	X	120,-	0
BrandA	$X+1$	110,-	-1

Sentiment Type 1: Initial load

Message scores

Search string	ID	Sentiment
BrandA	1	**-1**
BrandA	2	**-1**

Data Mart Type 1

Brand	Date	Net sales	Sentiment
BrandA	X	110,-	-1
BrandA	$X+1$	110,-	-1

Sentiment Type 2: proactive versioning

Message scores

Search string	ID	Version	Sentiment
BrandA	1	**1**	**0**
BrandA	2	**2**	**-1**

Data Mart Type 2

Brand	Date	**V-id**	Net sales	Sentiment
brandA	X	**a**	120,-	0
brandA	$X+1$	**b**	110,-	-1

Version dimension of Data Mart Type 2

V-id	Net sales version	Sentiment version
a	1	1
b	2	2

Sentiment Type 3: lazy amendment

At day $X+1$, the sample data mart schema changed: (i) new net sales measure due to the internal booking of discount vouchers; (ii) new sentiment measure due to changed opinion word list (see Section 2).

Message scores

Search string	ID	Sentiment	SentV2
BrandA	1	**0**	**-1**
BrandA	2	**-1**	**-1**

Data Mart Type 3

Brand	Date	N.s. old	Net sal.	Sent. old	Sent.
brandA	X	120,-	110,-	0	-1
brandA	$X+1$	110,-	110,-	-1	-1

Figure 6: Sample scenario illustrating SCM Types 0 to 3 at day X+1

only be used if the older values are irrelevant. For instance, if an important opinion word was overlooked for a sentiment measurement function, and thus all sentiments should be re-scored, SCM type 1 is an appropriate choice.

Type 1 SCMs avoid any growth of the hypercube schema and data when changes occur, but the re-computation of the measure for pre-existing business events is rather expensive, and implies operational data retention. The operational system(s) or the DW staging area must store all necessary data for long enough, instead of deleting the data upon a successful ETL cycle. If no such guarantee is possible, DW designers can revert to SCMs of Type 2, described next.

The top right of Figure 6 illustrates the effects of defining the sample measures as Type 1 SCMs. Under the initial load strategy of Type 1 all measure values obtained before day X are re-computed and overwritten at day X+1. As a result, the net sales measure at day X decreases, and the sentiment measure of message ID 1 is corrected from 0 to -1. Consequently, with SCM Type 1 the two problems discussed in Section 2 are solved—all measure values are comparable, yet the historical values become inaccessible.

3.3 Type 2: Proactive Versioning

SCMs of Type 2 account for *probable changes* of measure semantics in proactive manner by flagging each such measure within a dedicated version dimension that is added to the cube schema. The numerical *version flag* tracks all semantic changes of its allocated measure over time. Every change in the measure definition increases its version flag and the surrogate key of the version dimension, but leaves the pre-existing facts untouched. The version flag helps OLAP analysts to easily recognize incomparable measures. ETL procedures always employ the most recent measurement functions to newly inserted facts.

The main advantage of Type 2 SCMs is their simplicity. Apart from the small additional dimension table introduced at design time, the proactive versioning strategy avoids future schema updates. When measure definitions are changed, only an additional version tuple is created. Also it is unnecessary to retain any operational data for later ETL cycles as measures are never re-computed. This means that Type 2 SCMs can be implemented even in the most basic DW systems (e.g., without a staging area).

Using Type 2 SCMs in DW cubes, however, slightly reduces the flexibility of OLAP analysis. Specifically, the roll-up of the version flag is forbidden. Introducing a new version of a measure implies changing its meaning, so the measures remain comparable only within partitions that share the same version flag. Without previous conversion, aggregating measures across various versions is like aggregating monetary amounts of different currencies—which would clearly render a useless number. Furthermore, OLAP queries and reports must refer to the version flag(s), which slightly impedes query response time with an additional join. Physical partitioning of the fact table according to the version flag(s) is an easy solution to compensate for this disadvantage, though. SCM Type 2 is best used if the historical state of a measure is significant, but multi-version queries comparing several versions of the measure are unnecessary.

The bottom left of Figure 6 illustrates the two sample measures as Type 2 SCMs. Upon the semantic changes at day X, the proactive versioning strategy of Type 2 retains all previously obtained measure values. At day X+1, the version flag for both measures is increased, the new flags are recorded under a new surrogate key in the version dimension (version ID b), and the ETL process is adapted to the new measurement functions. Consequently, all measure values remain identical to those of the Type 0 SCMs (see top left of Figure 6), but now the version flag indicates the changed measure semantics from day X+1 onwards. The two problems mentioned in Section 2 were solved, yet this time no multi-version queries are possible—i.e., the measure values are logically partitioned by their version flags, and cannot be summarized across the different versions.

3.4 Type 3: Lazy Amendment

SCMs of Type 3 preserve the full history of measures by amending a separate measure attribute to the hypercube on every change of the measure definition. To minimize the impact on OLAP queries and reports, the new measure definition takes the old attribute name, whereas the older values are retained under the new attribute name. Since at design time no precaution is taken for future changes of the DW schema, we denote the strategy behind Type 3 SCMs as *lazy* amendment. As far as the data from transactional systems is still available, the new measurement function can be evaluated retrospectively for the pre-existing facts; otherwise the values remain empty. ETL procedures may evaluate several measure definitions for the newly inserted facts, or leave some older measures undefined, according to the DW designer's choice or the analysts' requirements.

Clearly, Type 3 is the most heavyweight design solution for evolving measures because each change entails a considerable administrative overhead. First, the retrospective computation of the new measure for the pre-existing business events in retained operational data is rather expensive. Second, even though the newer measure definition is referred by the old measure attribute, the update of the DW schema may induce a re-design of the OLAP queries and reports if more than one measure definitions are needed.

On the positive side, Type 3 SCMs are the most powerful design solution as they support multi-version queries in a straightforward way. By retaining the older measure definitions, the business analysts may have the same business events scored by various measurement functions. This can be useful, for instance, if a well-established sentiment measurement function for social media postings is to be complemented by an experimental sentiment analysis algorithm.

Hence, DW designers should adhere to the following three guidelines when applying the Type 3 of SCMs. First, use Type 3 only if the measure definition is to be changed rarely. Second, guarantee that the new measurement function can be evaluated for all pre-existing facts. To this end, retain the data from operational systems that fed the measure computation in previous ETL cycles. Third, check whether multi-version queries of the measure produce meaningful OLAP analysis results. SCM Type 3 is best used if the analysts can benefit from comparing two or more versions of the same measure. If such multi-version analysis is useless or irrelevant, it is better to revert to one of the other SCM types.

The bottom right of Figure 6 illustrates the sample measures as Type 3 SCMs. At day X+1, two additional measure columns are amended to the DW schema to account for the changed definitions of both, the net sales and the sentiment measure. The lazy amendment strategy of Type 2 retains all measure values computed until day X in the new columns named N.s. old and Sent. old. Furthermore, the changed measurement functions of net sales and sentiment are calculated retrospectively for all business events that occurred until day X in order to fill the new measure definitions, enabling multi-version querying. In the example we assume that the old measurement functions should be retained for both measures. Hence, the business analysts can use two complementary versions of both measures in OLAP analysis. Alternatively, leaving all the older measure columns empty from day X+1 onwards would question the Type 3 SCM design choice, as explained above.

3.5 Summary and Discussion

In the previous subsections we detailed various DW design options that enable an appropriate handling of changing measure definitions. The proposed solutions, denoted as SCM Types 0 to 3, require careful analysis of foreseeable changes to measure semantics at design time, but minimize the effort at run time when changes occur. Figure 7 below visualizes the sample net sales measure presented in Figure 6. The bottom right of Figure 7 highlights how the coexistence of various definitions of the same measure enables multi-version OLAP queries.

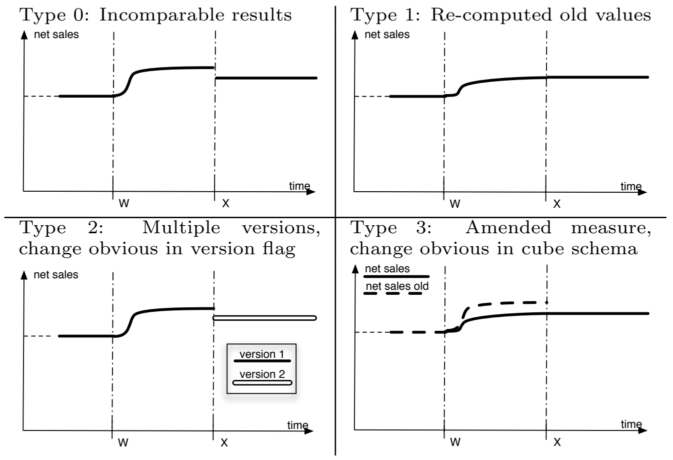

Figure 7: Net sales as SCM – (W) new marketing concept, (X) change of measurement function

Our examples have shown that SCMs allow to document changes in measure definitions mostly at the instance level with minimum design effort. Type 0, unless following the *conscious* do-nothing strategy, should be avoided as it leads to incomparable measures that compromise OLAP analysis. Type 1, the initial load strategy, re-computes and overwrites outdated measure values. Obviously, this is only appropriate if neither the history of measures, nor multi-version queries are needed. Furthermore, Type 1 SCMs require operational data retention which is often too costly. Type 2, the proactive versioning strategy, introduces a version flag for measures at design time. This solution augments the DW schema with a dedicated version dimension that preserves version flags of all measure definitions. Finally Type 3, the lazy amendment strategy, extends the DW schema with an additional measure attribute upon every measure definition update. Clearly the latter is the most heavyweight solution, and every schema update might affect also the OLAP queries and reports.

Remarkably, both the Type 2 and 3 SCMs allow a full historical track of measure definitions and values without a fully temporal model. Type 2 SCMs proactively account for potential changes, and thus handle even frequent measure re-definitions efficiently. In contrast, each re-definition of Type 3 SCMs changes the physical DW schema, and therefore potentially induces the re-design of OLAP queries and reports as well. Type 2 is the more flexible strategy, and DW designers should only use Type 3 SCMs if infrequent changes to measure definitions can be expected.

The overhead generated by the proposed SCM design solutions in terms of query response time and storage space is negligible when following our design guidelines in Table 1. Obviously Type 0 and Type 1 cause no additional query or storage costs. Type 2 requires an additional dimension table, which costs an extra join operation. This disadvantage can be compensated by physically partitioning the facts according to the version flag. The storage costs of the Type 2 version dimension will be negligible even for frequent updates. Finally, Type 3 with frequent updates involves a significant growth of the fact table, and also increased response time for multi-version queries. Hence, as stated above, Type 3 SCMs are only advisable if measure definitions are rarely updated, and if multi-version OLAP queries are meaningful.

4. RELATED WORK

The multi-dimensional model is prevalent for conceptual modeling of DW and OLAP. In the multi-dimensional model, the measures of facts are typically regarded as stable, especially the dimensional context related to the facts [5, 7, 14]. New fields of research such as Big Data or sentiment analysis have questioned this simplifying assumption. Also due to the constantly evolving application domains, in practice it is often necessary to adapt DW models over time. Driven by practical requirements, particularly those of complex measurement functions that we sketched in Section 2, several approaches perceive measures as a more flexible concept.

Temporal data warehousing systems introduce a valid time span for the objects in the DW schema, and thus support version changes of the DW data. In this respect, facts and dimensions must be treated fundamentally different. Since facts record measures on business events, fact data naturally conforms to the DW version valid when the event occurred. In the standard case of non-changing events, the time dimension straightforwardly gives their valid time. Dimensions, in contrast, represent *state*-oriented data, and thus the dimension members typically span valid time intervals [7, 11].

The *late registration* of business events is a special case in temporal data warehousing that leads to potentially error-prone, yet correctable measure values. In some domains with exceptionally long loading delays in ETL, business events may change after their original registration in the DW, when subsequent ETL cycles adjust the measure values of facts [6]. Similar problems occur in dedicated *real-time* and *zero-latency* DW systems [12]. Overwriting facts violates the *non-volatility* of DWs [8], but is manageable if the DW supports transaction time. Golfarelli & Rizzi showed that only bi-temporal semantics guarantees the accountability and traceability of historical measures under late measurements [6, 7]. In [6] they give design solutions for late registration of stock and flow measures that ensure sound OLAP analysis. In contrast to [6] where the authors study the consequences of evolving business events on OLAP accountability, the SCM approach assumes stable business events, normal delays in ETL, and rather manages changing measurement functions. The proposed SCM solutions are applicable to both measure semantics, stock and flow.

To achieve full temporal consistency of measures, Bruckner & Tjoa [3] model *revelation time* and *loading time* properties in addition to the more common valid time. In similar vein, the conceptual model of Malinowski & Zimányi [11] distinguishes even the four temporality types valid time, transaction time, life span, and loading time. As for our approach, SCMs of Types 2/3 support consistent semantics of measures, enabling sound OLAP queries even across several versions of a measure definition. Hence, a fully temporal model is unnecessary for SCMs.

Multi-version data warehousing systems (MV-DWs) even manage several full versions of the DW schema and data over time. Through their versioning approach MV-DWs re-establish the non-volatility property of DWs even in presence of schema changes. This implies that OLAP queries deliver the same results at any point in time. Hence, MV-DWs require access to the full historical track of DW schema versions. To this end, [5] introduce *augmented* DW schemas, whose key idea is to enrich previous versions of the DW schema with information of current modifications. The authors represent multi-dimensional schemas as graphs of simple functional dependencies, and define schema modifications in an elegant algebra of graph operations. Specifically, the approach enables flexible cross-version queries based on the augmented schemas. Other works in this field [2, 4, 15] define conceptual models together with schema evolution operators, and study consistency constraints for evolving schemata. The SCM Type 3 inherently supports cross-version queries by ensuring a common semantics of measures in all fact tuples, as mentioned above.

In temporal and multi-version DWs, measure definitions change implicitly on every update to one of its corresponding dimensions. This is clearly recognized by previous approaches [1–5, 7, 10, 15] , but most authors focus on changes in dimensions alone. Only Golfarelli & Rizzi [5, p. 455] explicitly address changing measure computation, albeit not as first-class modeling concept, but within the meta-data repository at manual human judgment. To the best of our knowledge, the only approach besides ours that includes changing measure definitions in conceptual DW modeling is the *temporal MultiDim* model [11]. The authors define a concise graphical notation with ER-like symbols and annotations of temporality types. Each of the SCMs defined in this paper can be expressed in the MultiDim model: Type 0 and Type 1 trivially as standard measure, with late measurements semantics in the latter case; Type 2 and Type 3 as measure with valid time (cf. Table 1). While the powerful features of MultiDim lead to rather complex DW models, the SCM approach can be used in standard DW models, and implemented in existing ROLAP platforms.

In contrast to [5, 11], this paper specifically investigates updates to measurement functions independent of the dimensional context. Our approach is appropriate for settings with evolving measure definitions but otherwise simple temporality semantics, such as in our example scenario. As demonstrated by our examples such updates have severe practical consequences, yet are too fine-grained for existing DW schema evolution operators. Without proper modeling, business analysts might not even notice the revised measure definitions, and reach false conclusions in OLAP analysis.

5. CONCLUSION

In this paper we investigated the problem of measure definitions in DW schemas that are subject to change. Current DW modeling approaches primarily consider changing dimensions, whereas only bi-temporal models or multi-version DWs support changing measures in a satisfying way. We introduced slowly changing measures (SCMs) as design solution that overcome this problem by documenting the changes in measure semantics. Most notably, SCMs are a lightweight approach compatible with standard DW models and technology. Table 1 recalls the main design guidelines for the four SCM types, and the strategies they follow.

Summarizing our results, SCMs manage changes in measure definitions with a minimum footprint in existing DW models. Changes in measure semantics are handled mostly at the instance level, avoiding major revisions of the physical DW schema. Type 0, unless consciously used, should in general be avoided as it leads to incomparable measures. Type 1 achieves comparable measures by re-computing and overwriting outdated measure values. Type 2 flags different measure definitions, leading to a logical fragmentation of the fact table by measure versions. Type 3 extends the DW schema with additional measures, which enables cross-version querying. It is striking that both, the Type 2 and 3 SCMs allow a full historical track of measure definitions and values without a fully temporal model, yet at the cost of some storage and query response time overhead.

In our examples we have shown how evolving measurement functions, unless properly handled, would render the fact table in an unsound state. Also we have demonstrated how to avoid incomparable measures with proper SCM modeling. SCMs give DW designers a set of design solutions that handle measure evolution with minimum additional design effort and standard ROLAP technology, and thus lead to more manageable DW models.

Acknowledgments

We thank Jean-Luc Dumont for key hints on reducing the complexity of the version dimension, and Stefan Anderlik for his valuable inputs to the related work section. Also we are grateful to the anonymous referees whose comments helped to considerably improve the presentation of our key ideas.

References

[1] Meenakshi Arora and Anjana Gosain. Schema evolution for data warehouse: A survey. *International Journal of Computer Applications*, 22(5): pp. 6–14, May 2011.

[2] Sandipto Banerjee and Karen C. Davis. Modeling data warehouse schema evolution over extended hierarchy semantics. *J. Data Semantics*, 13: pp. 72–96, 2009.

[3] Robert M. Bruckner and A. Min Tjoa. Capturing delays and valid times in data warehouses - towards timely consistent analyses. *J. Intell. Inf. Syst.*, 19(2): pp. 169–190, 2002.

[4] Johann Eder, Christian Koncilia, and Tadeusz Morzy. The COMET metamodel for temporal data warehouses. In *Proceedings of the 14th International Conference on Advanced Information Systems Engineering*, CAiSE 2002, Toronto, Canada, volume 2348 of *Lecture Notes in Computer Science*, pp. 83–99.

[5] Matteo Golfarelli, Jens Lechtenbörger, Stefano Rizzi, and Gottfried Vossen. Schema versioning in data warehouses: Enabling cross-version querying via schema augmentation. *Data Knowl. Eng.*, 59(2): pp. 435–459, 2006.

[6] Matteo Golfarelli and Stefano Rizzi. Managing late measurements in data warehouses. *International Journal of Data Warehousing and Mining*, 3(4): pp. 51–67, 2007.

[7] Matteo Golfarelli and Stefano Rizzi. A survey on temporal data warehousing. In John Erickson, editor, *Database Technologies: Concepts, Methodologies, Tools, and Applications*, pp. 221–237. IGI Global, 2009.

[8] William Inmon. *Building the Data Warehouse.* John Wiley & Sons, New York, 4th edition, 2005.

[9] Ralph Kimball. *The Data Warehouse Toolkit: Practical Techniques for Building Dimensional Datawarehouses.* John Wiley & Sons, New York, 2nd edition, 2002.

[10] Ziyang Liu, Bin He, Hui-I Hsiao, and Yi Chen. Efficient and scalable data evolution with column oriented databases. In *Proceedings of the 14th International Conference on Extending Database Technology*, EDBT/ICDT '11, New York, NY, USA, pp. 105–116.

[11] Elzbieta Malinowski and Esteban Zimányi. A conceptual model for temporal data warehouses and its transformation to the ER and the object-relational models. *Data Knowl. Eng.*, 64(1): pp. 101–133, 2008.

[12] Tho Manh Nguyen and A. Min Tjoa. Zero-latency data warehousing (ZLDWH): the state-of-the-art and experimental implementation approaches. In *Proceedings of the 4th International Conference on Computer Sciences: Research, Innovation and Vision for the Future*, RIVF 2006, Ho Chi Minh City, Vietnam, pp. 167–176.

[13] Tho Manh Nguyen, A. Min Tjoa, Jaromir Nemec, and Martin Windisch. An approach towards an event-fed solution for slowly changing dimensions in data warehouses with a detailed case study. *Data Knowl. Eng.*, 63(1): pp. 26–43, 2007.

[14] Oscar Romero and Alberto Abelló. Multidimensional design methods for data warehousing. In David Taniar and Li Chen, editors, *Integrations of Data Warehousing, Data Mining and Database Technologies*, pp. 78–105. Information Science Reference, 2011.

[15] Robert Wrembel and Bartosz Bebel. Metadata management in a multiversion data warehouse. *J. Data Semantics*, 8: pp. 118–157, 2007.