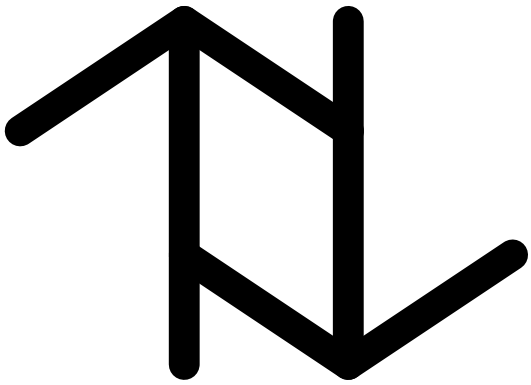

Specification-Based Data Reduction in Dimensional Data Warehouses

Janne Skyt, Christian S. Jensen, and Torben Bach Pedersen

TR-61

A TIMECENTER Technical Report

Title	Specification-Based Data Reduction in Dimensional Data Warehouses
Author(s)	Janne Skyt, Christian S. Jensen, and Torben Bach Pedersen
Publication History	July 2001. A TIMECENTER Technical Report.

TIMECENTER Participants

Aalborg University, Denmark
Christian S. Jensen (codirector), Michael H. Böhlen, Heidi Gregersen, Dieter Pfoser, Simonas Šaltenis, Janne Skyt, Giedrius Slivinskas, Kristian Torp

University of Arizona, USA
Richard T. Snodgrass (codirector), Dengfeng Gao, Vijay Khatri, Bongki Moon, Sudha Ram

Individual participants
Curtis E. Dyreson, Washington State University, USA
Fabio Grandi, University of Bologna, Italy
Nick Kline, Microsoft, USA
Gerhard Knolmayer, Universty of Bern, Switzerland
Thomas Myrach, Universty of Bern, Switzerland
Kwang W. Nam, Chungbuk National University, Korea
Mario A. Nascimento, University of Alberta, Canada
John F. Roddick, University of South Australia, Australia
Keun H. Ryu, Chungbuk National University, Korea
Michael D. Soo, amazon.com, USA
Andreas Steiner, TimeConsult, Switzerland
Vassilis Tsotras, University of California, Riverside, USA
Jef Wijsen, University of Mons-Hainaut, Belgium
Carlo Zaniolo, University of California, Los Angeles, USA

For additional information, see The TIMECENTER Homepage:
URL: <http://www.cs.auc.dk/TimeCenter>

The TIMECENTER icon on the cover combines two "arrows." These "arrows" are letters in the so-called *Rune* alphabet used one millennium ago by the Vikings, as well as by their precedessors and successors. The Rune alphabet (second phase) has 16 letters, all of which have angular shapes and lack horizontal lines because the primary storage medium was wood. Runes may also be found on jewelry, tools, and weapons and were perceived by many as having magic, hidden powers.

The two Rune arrows in the icon denote "T" and "C," respectively.

Abstract

Many data warehouses contain massive amounts of data and grow rapidly. Examples include warehouses with retail sales data capturing customer behavior and warehouses with click-stream data capturing user behavior on web sites. The sheer size of these warehouses makes them increasingly hard to manage and query efficiently. As time passes, old, detailed data in the warehouses tend to become less interesting. However, higher-level summaries of the data, taking up far less space, continue to be of interest. Thus, it is possible to perform data reduction in dimensional data warehouses by aggregating data to higher levels in the dimensions.

This paper presents an effective technique for data reduction that handles the gradual change of the data from new, detailed data to older, summarized data. The technique enables huge storage gains while ensuring the retention of essential data. The data reduction is based on formal specifications of when data should be aggregated to higher levels. Care is taken to ensure that the irreversible data reductions are without semantic problems. It is defined how queries over the resulting data with varying levels of detail are handled, and a strategy for implementing the technique using standard data warehouse technology is described.

Key words and phrases: Data reduction, data warehousing, multidimensional data models, physical deletion

1 Introduction

Many of today's data warehouses are very large and continue to be updated almost continuously or bulk-loaded regularly with new data. Such data warehouses occur in many lines of business, including retail, finance, telecommunication, and health care, to name but a few.

Recently, the drive to analyze, understand, and react upon user behavior on web sites has led to the appearance of many very large click-stream warehouses. Common to these is that their sheer size, often on the order of terabytes, is making it hard to manage and query them with the desired efficiency. Fortunately, much of the data in these warehouses becomes interesting only at an aggregated level as the data ages, i.e., the detailed data gradually loses its value as it gets older. This offers a possibility for huge storage savings, by storing data only at the desired level of detail.

Data warehouse systems are generally based on a dimensional view of data, in which *business facts* with associated measures are characterized by descriptive values drawn from a number of *dimensions*; and the values of a dimension are typically organized in a containment-type hierarchy. A prototypical query applies an aggregate function, such as sum, to a measure associated with some facts that are characterized by specific values from the dimensions. The level of detail of a fact is then given by the combination of levels that the fact's measure is aggregated to in the different dimensions, ranging from bottom levels (the most detailed) to top levels (aggregated over the whole dimension).

This paper presents a powerful and easy-to-use technique for aggregation-based data reduction that enable the gradual change of the data from being detailed to being aggregated. The technique provides the means to realize huge storage gains while still retaining the data that users see as essential. The data reduction is achieved by specifying criteria for when data should be aggregated to higher levels in the dimensions. For example, sums of sales should be aggregated from the daily to the monthly level when between six months and three years old, and further to the yearly level when more than three years old. Criteria and evaluation mechanisms are given that ensure sound and meaningful data reduction specifications, so that semantic problems are avoided. It is shown how queries are applied to reduced data, with possibly varying levels of detail, and a strategy for implementing the technique effectively using standard technology is described.

We believe this paper to be the first to present techniques for allowing multidimensional data to be gradually and automatically reduced, e.g., over time, by repeatedly aggregating them based on specifications.

We also believe to be the first to consider the issues related to the semantic correctness of a gradual data reduction process. Additionally, we have found no account in the literature for the querying of multidimensional data that is gradually being aggregated.

Related work exist in several areas. In the context of relational databases with transaction-time support [14] some work has been reported on the physical deletion of rows based on formal specifications [16]. Another line of work [15], also in the context of the relational model, proposes a kind of view that is unaffected by deletions on the underlying relations as opposed to materialized views [8, 17], and is sensitive towards updates as opposed to snapshots [1]. In the context of warehousing Garcia-Molina et al. [6] explore how to "expire" (delete) data from materialized views so that a set of predefined, regular views on these materialized views are unaffected and can be maintained consistently with future updates. In comparison, this paper considers *aggregation* of data, retaining the essential information in the data warehouse while reducing the storage requirements considerably. Also, this paper takes into account multidimensional concepts, such as measures and dimensions with hierarchies. The work on dimensionality reduction in data warehouses [10] considers the removal of entire dimensions from all the data, based on the importance of the dimensions for the measure values, e.g., the sum of sales. In contrast, our work achieves a reduction in data volume, while still allowing all dimensions to remain in the warehouse, by offering mechanisms for selectively aggregating data to higher levels. Previous work has also dealt with general techniques for data reduction [2] such as wavelets, sampling, and aggregation. These types of work differ in focus from ours, which offers rule-based specifications for when data reduction, in our case using aggregation, should take place, allowing the reduction process to be automatic and transparent to the user. The aggregation approach to data reduction has the advantage that it, in contrast to techniques such as wavelets and sampling, keeps the *exact* values for higher-level aggregates, which is likely to be important in a data warehouse.

The remainder of the paper is structured as follows. Section 2 presents a short example, to be used for illustration throughout the paper. Section 3 formally presents the multidimensional data model. Section 4 defines the syntax and semantics of the data reduction specification, and Section 5 covers the issues that arise when changing a data reduction specification. Section 6 is devoted to the querying of a reduced data warehouse, while implementation is the topic of Section 7. Finally, Section 8 concludes and points to future research topics. Appendix A presents the data and formalism corresponding to the example in Section 2.

2 Illustrating Example

This section presents a multidimensional object to be used for illustration throughout the paper, namely the database of an Internet Service provider (ISP). This database accumulates click-streams and is used for analysis of user behavior. In this database, each click entered by a user is registered with its occurrence time, target_url, and other kinds of information. To keep the illustrations readable, the example considers only facts (clicks) and two dimensions (URL, $Time$). In Appendix A we present the tables of example data as well as the full formalization of the multidimensional object.

In the schema used, the URL and $Time$ dimensions have the following hierarchies of category types.

$$url \leq_{URL} domain \leq_{URL} domain\_grp \leq_{URL} \top_{URL} \quad (1)$$

$$day \leq_{Time} week \leq_{Time} \top_{Time} \text{ and } day \leq_{Time} month \leq_{Time} quarter \leq_{Time} year \leq_{Time} \top_{Time} \quad (2)$$

The two orderings indicate containment. For example, $day \leq_{Time} month$ means that each element in category type day is contained in an element of category type $month$. Figure 1 illustrates this containment using the sample data used throughout the paper. The dimension values are related by bold lines in tree structures. Note, that the $Time$-dimension, which has parallel hierarchies, is a graph with multiple connections from leaf to root.

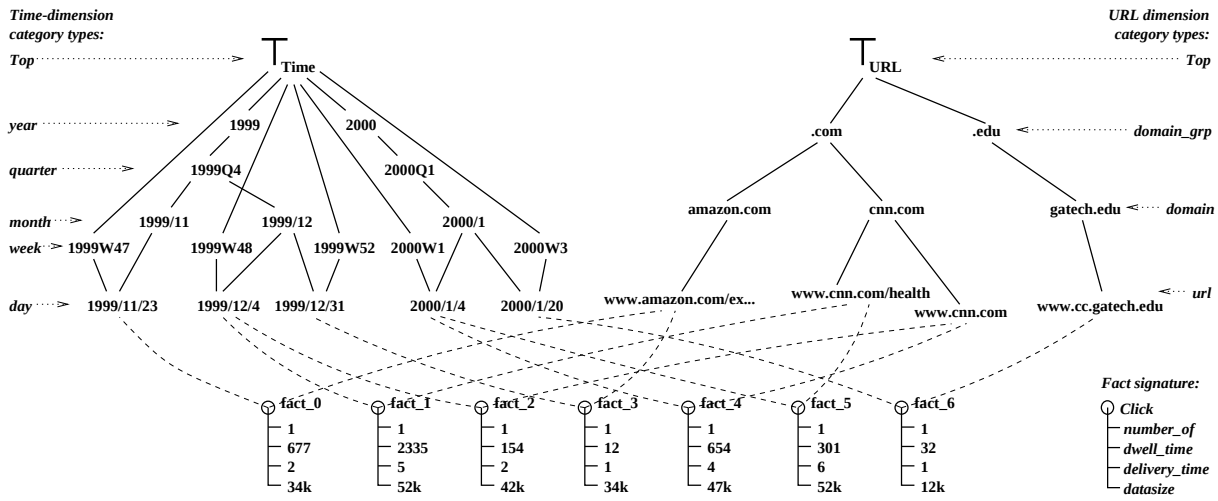

Figure 1: Graphical Illustration of Example MO

Each click fact refers to one bottom-level category type for each dimension, i.e., to a day-value and a url-value. Note, that dotted lines denote relationships between a fact and its dimension values. Also each click fact refers to the measures $Number\_of$, $Dwell\_time$, $Delivery\_time$, and $Datasize$. In the figure, this is captured by the fact signature.

Being typical, our service provider registers a very large number of clicks per day, leading to an extremely fast growing fact table. As they age, the interest in the individual click facts decreases, one reason being that they tend to become inaccurate. For example, web pages and URLs change over time, as do users. Such changes tend to result in facts being related to inaccurate dimension values. This provides the impetus for applying the type of data reduction proposed in the remainder of this paper.

3 Data Model

This section defines a prototypical multidimensional data model to be used as the concrete context for specifying the paper's data reduction mechanisms. The model precisely and concisely captures core multidimensional concepts such as categories, dimensions, and automatic aggregation.

We consider first the schema of a multidimensional object, then the instances of the schema. An *n-dimensional fact schema* is a three-tuple $\mathcal{S} = (\mathcal{F}, \mathcal{D}, \mathcal{M})$, where $\mathcal{F}$ is a *fact type*, $\mathcal{D} = \{\mathcal{T}_i \mid i \in 1, \ldots, n\}$ is its corresponding *dimension types*, and $\mathcal{M} = \{\mathcal{M}_j \mid j \in 1, \ldots, m\}$ is its *measure types*. The fact type and the measure types are *names*.

EXAMPLE: In our example in Section 2, $Click$ is the fact type and $Number\_of$, $Dwell\_time$, $Datasize$ and $Delivery\_time$ are the measure types. The dimension types are treated next. □

A dimension type $\mathcal{T}$ is a four-tuple $(\mathcal{C}, \leq_\mathcal{T}, \top_\mathcal{T}, \bot_\mathcal{T})$, where $\mathcal{C} = \{\mathcal{C}_j \mid j \in 1, \ldots, k\}$ are the *category types* of $\mathcal{T}$, $\leq_\mathcal{T}$ is a partial order on the $\mathcal{C}_j$'s, with $\top_\mathcal{T} \in \mathcal{C}$ and $\bot_\mathcal{T} \in \mathcal{C}$ being the top and bottom element of the ordering, respectively. The intuition is that one category type is "greater than" another category type if each member of the former's extension logically contains several members of the latter's extension, i.e., they have a larger element size. The top element of the ordering corresponds to the largest possible element size, that is, there is only one element in its extension, logically containing all other elements. We say that $\mathcal{C}_j$ *is a category type of* $\mathcal{T}$, written $\mathcal{C}_j \in \mathcal{T}$, if $\mathcal{C}_j \in \mathcal{C}$. We assume a function $Anc : \mathcal{C} \mapsto 2^\mathcal{C}$ that gives the

set of immediate ancestors of a category type $\mathcal{C}_j$. We say that the hierarchy in a dimension type $\mathcal{T}$ is *linear* if $\leq_{\mathcal{T}}$ is total. Otherwise, the hierarchy is *non-linear*.

EXAMPLE: URLs are contained in domains, which are contained in domain groups. Thus, the URL dimension type has the following order on its category types: $\perp_{URL} = url < domain < domain\_grp < \top_{URL}$, and, e.g., $Anc(domain) = \{domain\_grp\}$. Using the formalism, the schema of the multi-dimensional object in Figure 1 is written as follows. (The fully formalized MO for the example is presented in Appendix A.)

$$\mathcal{S} = (\mathcal{F}, \mathcal{D}, \mathcal{M}),\ \mathcal{F} = Click,\ \mathcal{D} = \{Time, URL\},\ \mathcal{M} = \{Number\_of, Dwell\_time, Delivery\_time, Datasize\}$$
$$Time = (\{day, week, month, quarter, year\}, \leq_{Time}, \top_{Time}, \perp_{Time} = day),$$
$$day \leq_{Time} month \leq_{Time} quarter \leq_{Time} year \leq_{Time} \top_{Time} \text{ and } day \leq_{Time} week \leq_{Time} \top_{Time}$$
$$URL = (\{url, domain, domain\_grp\}, \leq_{URL}, \top_{URL}, \perp_{URL} = url),$$
$$url \leq_{URL} domain \leq_{URL} domain\_grp \leq_{URL} \top_{URL}$$

The hierarchy in the URL dimension type is linear, while the hierarchy in the $Time$ dimension type is non-linear. □

A *category* C_j of type $\mathcal{C}_j$ is a set of *dimension values* v. A *dimension* D of type $\mathcal{T} = (\{\mathcal{C}_j\}, \leq_{\mathcal{T}}, \top_{\mathcal{T}}, \perp_{\mathcal{T}})$ is a two-tuple $D = (C, \leq_D)$, where $C = \{C_j\}$ is a set of categories C_j such that $Type(C_j) = \mathcal{C}_j$ and $\leq_D$ is a partial order on $\cup_j C_j$, the union of all dimension values in the individual categories.

The partial order is defined as follows. Given two values v_1, v_2 then $v_1 \leq_D v_2$ if v_1 is logically contained in v_2, i.e., v_2 can be considered as a set containing v_1. We say that C_j is a category of D, written $C_j \in D$, if $C_j \in C$. For a dimension value v, we say that v is a dimension value of D, written $v \in D$, if $v \in \cup_j C_j$.

We assume that the partial order $\leq_{\mathcal{T}}$ can also be used directly on the categories. The category $\perp_D$ in dimension D contains the values with the smallest value size. The category with the largest value size, $\top_D$, contains exactly one value, denoted $\top$. For all values v of D, $v \leq_D \top$. Value $\top$ is similar to the *ALL* construct of Gray et al. [7]. We assume that the partial order on category types and the function Anc work directly on categories, with the order given by the corresponding category types.

We say that the dimension $D' = (C', \leq_{D'})$ is a *subdimension* of the dimension $D = (C, \leq_D)$ if $C' \subseteq C$ and $v_1 \leq_{D'} v_2 \Leftrightarrow \exists C_1, C_2 \in C'(v_1 \in C_1, v_2 \in C_2 \wedge v_1 \leq_D v_2)$, that is, D' is a subset of the set of categories of D and $\leq_{D'}$ is the restriction of $\leq_D$ to these categories. We note that D is a subdimension of itself.

EXAMPLE: We obtain a subdimension of the URL dimension from the previous example by removing the url and $domain$ categories, retaining only the categories $domain\_grp$ and $\top_{URL}$. □

A *fact* f of type $\mathcal{F}$ is an object with a unique identity such that $Type(f) = \mathcal{F}$. Let F be a set of facts, and $D = (\{C_j\}, \leq_D)$ a dimension. A *fact-dimension relation* between F and D is a set $R = \{(f, v) \mid f \in F \wedge v \in \cup_j C_j\}$. Thus R links facts to dimension values. We say that fact f is *characterized by* dimension value v, written $f \rightsquigarrow v$, if $\exists v_1 \in D\ ((f, v_1) \in R \wedge v_1 \leq_D v)$.

Note that facts can be characterized by dimension values in several dimension categories. The intuition is that a fact is characterized by all dimension values that are either directly (through a fact-dimension relation) or indirectly related to it (the value is an ancestor of a directly related value). We require that $\forall f \in F\ (\exists v \in D((f, v) \in R))$; thus we do not allow missing values. The reasons for disallowing missing values are that they complicate the model and often have an unclear meaning. If it is unknown which dimension value a fact f is characterized by, we add the pair $(f, \top)$ to R, thus indicating that we cannot

characterize f within the particular dimension. Also, we require that facts are only mapped to *one* dimension value in each dimension, as is standard in multidimensional models. Finally, we require that facts inserted by users are mapped to dimension values in bottom categories. The more general capability of the model to accommodate the mapping of facts to dimension values in any category will be exploited by the data reduction facilities.

Let F be a set of facts and T a domain. A *measure* M for F and T, of type $\mathcal{M}_M$, is a function $M : F \mapsto T$ such that $\mathit{Type}(M) = \mathcal{M}_M$. A measure M has associated with it a *default aggregate function* $a_M : T \times T \mapsto T$ (note that we can also naturally use a_M on multi-sets). The default aggregate function must be distributive.

EXAMPLE: In the example we have four measures, Number_of, Dwell_time, Delivery_time, and Datasize. The default aggregation function for all four is SUM. □

A *multidimensional object* (MO) is a five-tuple $O = (\mathcal{S}, F, D, R, M)$, where $\mathcal{S} = (\mathcal{F}, \mathcal{D} = \{\mathcal{T}_i\}, \mathcal{M} = \{\mathcal{M}_j\})$ is the fact schema, $F = \{f\}$ is a set of *facts* f where $\mathit{Type}(f) = \mathcal{F}$, $D = \{D_i \mid i \in 1, \ldots, n\}$ is a set of *dimensions* where $\mathit{Type}(D_i) = \mathcal{T}_i$, $R = \{R_i \mid i \in 1, \ldots, n\}$ is a set of fact-dimension relations, such that $\forall i \in 1, \ldots, n((f, v) \in R_i \Rightarrow f \in F \wedge \exists C_j \in D_i(v \in C_j))$, and $M = \{M_j \mid j \in 1, \ldots, m\}$ is a set of *measures* such that $\mathit{Type}(M_j) = \mathcal{M}_j$.

4 Data Reduction in Multidimensional Objects

This section describes the actual data reduction technique.

When reducing the size of a multidimensional object, the most promising direction is to reduce the number of facts since facts typically take up 95% of the total data warehouse storage. A reduction in the number of facts may be accomplished by simply deleting facts. However, we want to support the retention of useful higher-level information, so instead we propose to selectively aggregate facts to higher levels of granularity, followed by physical deletion of the lower-level facts. We also allow for progressive aggregation as time passes. The specification of the reduction is given by a number of reduction *actions*, each of which specifies that a certain range of data must be aggregated to certain dimension levels.

A data reduction system has certain natural characteristics. The aggregation of a fact to a certain level is inherently *irreversible*, for which reason the set of actions must continue to specify aggregation of the fact to at least the same level. The actions should be seen as a *set*, i.e., there is no ordering on the individual actions and their effect is independent of when they were introduced into the system. Any action is permitted in the system if it *does not violate consistency and irreversibility*. As a consequence of this, we allow "useless" actions that specify aggregation to levels below the levels specified by other actions, as this is sometimes useful. Also, for any fact in a reduced MO, it is important to be able to determine the specific action that *caused* the fact to be aggregated to its current level, e.g., to communicate to users why data is aggregated the way it is.

4.1 Data Reduction Specification Notation

We proceed to define the notation and pertinent conventions for data reduction actions.

Specifically, we let ρ denote the *action type*; data reduction actions are then instances of this type. An action a_i aggregates certain facts to some higher level of granularity, then removes the facts. Auxiliary function Cat_i takes an action specification as argument and returns the category type $\mathcal{C}_{ij}$ to which the action will aggregate facts in the i'th dimension. Similarly, Cat takes an action as argument and returns the n-tuple of category types $\mathcal{C}_{ij_i}$ to which the action will aggregate the facts to in the n dimensions. A data reduction specification is defined as follows.

DEFINITION 1 *A data reduction specification V is a two-tuple $(A, \leq_{\mathcal{V}})$, where $A = \{a_i \mid i \in 1, \ldots, k\}$ is a set of action specifications, and $\leq_{\mathcal{V}}$ is a partial order on the action specifications, defined as follows.*

$$a_1 \leq_{\mathcal{V}} a_2 \iff^{def} \forall i \in 1, \ldots, n\,(Cat_i(a_1) \leq_{\mathcal{T}_i} Cat_i(a_2)) \tag{3}$$

□

Next, action specifications must obey the syntax given in Table 1 and must also satisfy usual conventional semantic constraints, to be specified shortly. In the table, α and σ denote the aggregation and selection operators on multidimensional objects (to be defined in Section 6). Also, $C_{Time\ j_{Time}}$ and C_{ij_i} are categories in the $Time$-dimension and the i'th dimension, respectively, and Obj is a multidimensional object.

a	::=	$\rho(\alpha[Clist]\ \sigma[Pexp]\ (Obj))$
$Clist$	::=	$\mathcal{C}_{ij} \mid \mathcal{C}_{ij}, Clist$
$Pexp$	::=	P bop $P \mid (P) \mid \neg P \mid P$
P	::=	$C_{Time\ j_{Time}}$ op $tt \mid C_{Time\ j_{Time}} \in \{tt, \ldots, tt\} \mid C_{ij_i}$ op $d \mid C_{ij_i} \in \{d, \ldots, d\} \mid$ true $\mid$ false
tt	::=	$tt - tt \mid tt + tt \mid (tt) \mid t \mid s$
bop	::=	$\wedge \mid \vee$
op	::=	$< \mid > \mid = \mid \leq \mid \geq \mid \neq$

Table 1: Syntax of Action Specifications

Let S be a finite non-empty set of time spans, i.e., unanchored time intervals (e.g., 2 days, 3 years). Then it is required that $t \in T \cup \{NOW\}$ and $s \in S$, where NOW is the time variable that will allow dynamic actions [4]. For the expression '$C_{Time\ j_{Time}}$ op tt' it is required that $Type(tt) = \mathcal{C}_{Time\ j_{Time}}$ and that the operator 'op' should be defined for elements of type $\mathcal{C}_{Time\ j_{Time}}$, similarly, it is required that $Type(d) = \mathcal{C}_{ij_i}$ and that 'op' is defined for elements of this type. Following normal conventions, we allow ourselves to use expressions that are equivalent to those strictly generated by the grammar. For example, we will use $tt_1 < tt_2 < tt_3$ instead of $tt_1 < tt_2 \wedge tt_2 < tt_3$. We require that selection predicates be in disjunctive normal form (DNF).

Next, the $Clist$ of an action must contain exactly one category type from each dimension of the MO being reduced. These denote the granularity achieved by the action. Also, if there is a predicate on dimension D_i in the predicate $Pexp$, limiting dimension values of the category type $\mathcal{C}_{ij_{Pexp}}$ in this dimension, then the category type $\mathcal{C}_{ij_{Clist}}$ mentioned in $Clist$ must obey the inequality $\mathcal{C}_{ij_{Clist}} \leq_{\mathcal{T}_i} \mathcal{C}_{ij_{Pexp}}$. This ensures that, for each dimension, an action will aggregate to a category not exceeding the one referred in its predicate, which ensures that the predicate can continuously be evaluated on the aggregated facts.

EXAMPLE: Assume the MO in Figure 1 and the data reduction specification $V = (A, \leq_{\mathcal{V}})$, and $A = \{a_1, a_2\}$ where

$$\begin{aligned} a_1 = \rho(\alpha[&Time.month, URL.domain]\ \sigma[URL.domain\_grp = \texttt{.com} \wedge \\ &NOW - 12\ months < Time.month \leq NOW - 6\ months](O)) \end{aligned} \tag{4}$$

$$\begin{aligned} a_2 = \rho(\alpha[&Time.quarter, URL.domain] \\ &\sigma[URL.domain\_grp = \texttt{.com} \wedge Time.quarter \leq NOW - 4\ quarters](O)) \end{aligned} \tag{5}$$

For example, action a_2 specifies aggregation to the granularities $Time.quarter$ and $URL.domain$. This means that all facts characterized by the same $domain$-value in the URL dimension, and by the same $quarter$-value in the $Time$ dimension will be aggregated to a single new fact (see Section 6.3 for a specification of the aggregate formation operator.) Action a_2, however, only specifies the aggregation for facts

characterized by a $quarter$-value in the $Time$ dimension less than or equal to current time minus 4 quarters. As required, a_2 selects based on a level of granularity at least as high as the level it aggregates to.

We see that $a_1 \leq_{\mathcal{V}} a_2$, since a_1 aggregates to the granularity of $(Time.month, URL.domain)$, and since a_2 aggregates to $(Time.quarter, URL.domain)$, which is higher. For this specification, the ordering is total. Had there been a third action that aggregated to $Time.week$ and $URL.url$, the ordering would be partial. □

4.2 Auxiliary Functions

In order to define the semantics of data reduction, some auxiliary functions are needed. Specifically, it is convenient to have auxiliary functions that find the granularity specified by an action, the maximum granularity specified for a fact, and the resulting cell of dimension values for a fact. To define these functions, we first introduce some notation.

Throughout the section, we assume a multidimensional object $O = (\mathcal{S}, F, D, R, M)$ of n dimensions and a data reduction specification $V = (\{a_1, \ldots, a_k\}, \leq_{\mathcal{V}})$. A *cell* is an n-tuple of dimension values $(v_1, \ldots, v_n)$, one for each dimension.

The *ordering* $\leq_{\mathcal{D}}$ of two granularities $(\mathcal{C}_{1j_1}, \ldots, \mathcal{C}_{nj_n})$ and $(\mathcal{C}_{1k_1}, \ldots, \mathcal{C}_{nk_n})$ is defined as follows.

$$(\mathcal{C}_{1j_1}, \ldots, \mathcal{C}_{nj_n}) \leq_{\mathcal{D}} (\mathcal{C}_{1k_1}, \ldots, \mathcal{C}_{nk_n}) \stackrel{def}{\iff} \forall i \in 1, \ldots, n\ (\mathcal{C}_{ij_i} \leq_{\mathcal{T}_i} \mathcal{C}_{ik_i}) \tag{6}$$

Then, assuming that $\leq_{\mathcal{D}}$ is a total ordering for the input set, the function $\max_{\leq_{\mathcal{D}}} : 2^{\mathcal{C}_1 \times \cdots \times \mathcal{C}_n} \mapsto \mathcal{C}_1 \times \cdots \times \mathcal{C}_n$ returns the maximum granularity.

Let action a be given as "$\rho(\alpha[\mathcal{C}_{1j_1}, \ldots, \mathcal{C}_{ij_i}, \ldots, \mathcal{C}_{nj_n}]\sigma[P](O))$" and let t be the time when the predicate is evaluated. The three functions Cat_i, Cat, and $Pred$ are then defined as follows.

$$Cat_i(a) \stackrel{def}{=} \mathcal{C}_{ij_i} \tag{7}$$

$$Cat(a) \stackrel{def}{=} (\mathcal{C}_{1j_1}, \ldots, \mathcal{C}_{nj_n}) \tag{8}$$

$$Pred(a, t) \stackrel{def}{=} \{(v_1, \ldots, v_n) \mid (v_1, \ldots, v_n) \in D_1 \times \cdots \times D_n \wedge P[NOW \leftarrow t]\ (v_1, \ldots, v_n)\} \tag{9}$$

Thus, $Cat_i(a)$ returns the category type in the i'th dimension to which action a specifies aggregation, and $Cat(a)$ returns the n-tuple of category types representing the granularity to which a specifies aggregation.

The function invocation $Pred(a, t)$ returns the set of cells $(v_1, \ldots, v_n)$ of any granularity, such that any cell in the set satisfies the selection predicate P of action a at time t. The definition states that if the NOW-variable is used in predicate P, then it is replaced by time t. Thus, the n-tuples of dimension values containing the time values v_i satisfying this new predicate are in the set of $Pred$. This way, $Pred$ always captures the n-tuples of dimension values that currently (at time t) satisfy the predicate of action a.

EXAMPLE: Assume the MO presented in Section 2, action a_2 in (5), and let the current time be $2000/11/5$. Then the auxiliary functions above will give the following results.

$$Cat_{Time}(a_2) = Time.quarter,\ Cat(a_2) = (Time.quarter, URL.domain),$$
$$Pred(a_2, 2000/11/5) = \{(1999/11/23, \texttt{www.cnn.com}), (1999/12/4, \texttt{www.cnn.com}),$$
$$\ldots\ldots, (1999/11/23, \texttt{www.cnn.com/health}), \ldots\ldots, (1999, \texttt{www.amazon.com/exec/..})\}$$

Here, $Pred$ selects the cells with a $Time.quarter$-value that satisfies "$Time.quarter \leq 2000Q4 - 4$." □

Assume a fact f in the MO. Then we define the function $Gran$ that returns the current granularity of the fact f as an n-tuple of category types, i.e., for all values v_i to which f has a relation (f, v_i) in R_i, $Gran(f)$ returns the n-tuple of $Type(v_i)$'s. Note that only one such v_i exists for each dimension.

$$Gran(f) \stackrel{def}{=} (Type(v_1), \ldots, Type(v_n)) \qquad \text{such that } \forall i \in 1, \ldots, n\ (\exists v_i \in D_i\ ((f, v_i) \in R_i)) \quad (10)$$

Function $Spec\_gran(f, t)$ returns the set of granularities specified as aggregation levels for f at time t. Each granularity in this set is restricted to be specified by an action a such that if $v_1, \ldots, v_n$ is the cell that f maps to directly, then $v_1, \ldots, v_n$ satisfies the predicate of a at time t.

$$\begin{aligned} Spec\_gran(f, t) \stackrel{def}{=} \{(\mathcal{C}_{1j_1}, \ldots, \mathcal{C}_{nj_n}) \mid \\ \exists a \in A\ (\exists (v_1, \ldots, v_n) \in D_1 \times \cdots \times D_n\ ((v_1, \ldots, v_n) \in Pred(a, t) \wedge \\ \forall i \in 1, \ldots, n\ ((f, v_i) \in R_i\ \wedge\ Cat_i(a) = \mathcal{C}_{ij_i})))\} \cup \{Gran(f)\} \end{aligned} \quad (11)$$

In the definition, the granularity of f is added to the set to ensure that the set is non-empty. For all other granularities being returned, there must exist an action such that the dimension values mapped to f satisfy a's predicate.

Using the functions just defined, we define the function $Cell$ to return the cell of dimension values that a fact will be aggregated to at time t.

$$\begin{aligned} Cell(f, t) \stackrel{def}{=} (v'_1, \ldots, v'_n) \quad \text{such that} \\ \forall i \in 1, \ldots, n\ (v'_i \in D_i\ \wedge\ f \rightsquigarrow_i v'_i) \wedge (Type(v'_1), \ldots, Type(v'_n)) = \max_{\leq_{\mathcal{D}}}(Spec\_gran(f, t)) \end{aligned} \quad (12)$$

it follows that the cell of a fact f is an n-tuple of dimension values v'_i (one for each dimension) where f maps directly or indirectly to each dimension value. Having the granularity of f included in $Spec\_gran$ ensures that the granularity of the cell is the maximum granularity of f's own granularity and the granularities specified by actions with predicates satisfied by the dimension values of fact f.

EXAMPLE: Assume the MO in Figure 1 and specification $\{a_1, a_2\}$ (see specifications (4) and (5)). Then consider $fact\_1$, which maps to the values $1999/12/4$ and `www.cnn.com/health`, and the time $2000/11/5$. Then the auxiliary functions evaluate as follows.

$$\begin{aligned} &Gran(fact\_1) = (Time.day, URL.url) \\ &Spec\_gran(fact\_1, 2000/11/5) = \{(Time.day, URL.url), (Time.month, URL.url), \\ &\qquad\qquad (Time.quarter, URL.domain)\} \\ &Cell(fact\_1, 2000/11/5) = (1999Q4, \texttt{cnn.com}) \end{aligned}$$

Note that $Spec\_gran$ captures the granularities specified by actions a_1 and a_2, as well as the current granularity of $fact\_1$. $Cell$ uses the maximum granularity and returns the specific cell of dimension values of this granularity. □

The final auxiliary function needed is $AggLevel_i$, which returns the maximum aggregation level in the i'th dimension, given the dimension values of a cell $(v_1, \ldots, v_n)$ and a time point t.

$$\begin{aligned} AggLevel_i(v_1, \ldots, v_n, t) \stackrel{def}{=} \\ \max_{\leq_{\mathcal{T}_i}}(\{\mathcal{C}_{ij} \mid \exists a \in A\ (Cat_i(a) = \mathcal{C}_{ij} \wedge (v_1, \ldots, v_n) \in Pred(a, t))\} \cup \{\bot_{\mathcal{T}_i}\}) \end{aligned} \quad (13)$$

The definition states that given the cell of dimension values any action having a predicate satisfied by this cell at time t a category type $\mathcal{C}_{ij}$ is considered. Of these category types, the largest is returned by $AggLevel_i$. $\bot_{\mathcal{T}_i}$ will be the answer if no reduction is specified for the cell of $(v_1, \ldots, v_n)$.

4.3 Non Crossing and Growing Data Reduction

A data reduction specification must be $NonCrossing$ and $Growing$ to be semantically meaningful. Here, we define these two properties and explain why they are needed. We assume a multidimensional object $O = (\mathcal{S}, F, D, R, M)$ and a data reduction specification $V = (A, \leq_\mathcal{V})$ where $A = \{a_1, \ldots, a_k\}$.

First, actions in a specification must be $NonCrossing$. This means that actions with overlapping predicates cannot be allowed to aggregate to crossing granularities.

$$\begin{aligned} &NonCrossing(V) \iff^{def} \\ &\quad \forall a_1, a_2 \in A\ (\exists t \in T\ (Pred(a_1, t) \cap Pred(a_2, t) \neq \emptyset) \Rightarrow (a_1 \leq_\mathcal{V} a_2 \vee a_2 \leq_\mathcal{V} a_1)) \end{aligned} \tag{14}$$

Let actions a_j and a_k aggregate to granularities $(\mathcal{C}_{1j_1}, \ldots, \mathcal{C}_{nj_n})$ and $(\mathcal{C}_{1k_1}, \ldots, \mathcal{C}_{nk_n})$, respectively. If there is a time t where the predicates of a_j and a_k have intersecting value sets, one of a_j and a_k must be larger than the other according to $\leq_\mathcal{V}$. This means that there can be no two dimensions x and y such that a_j aggregates higher in the x'th dimension and a_k aggregates higher in the y'th dimension.

There are two main reasons for requiring this property. One is to guarantee that action predicates can be evaluated. The other is to be able to contend with non-linear hierarchies. These will be explained in examples next.

EXAMPLE: Using the ongoing example, assume the actions a_2 (equation 5) and a_3 (equation 15, below) at the current time of $2000/11/5$.

$$\begin{aligned} a_3 = \rho(\alpha[&Time.month, URL.domain\_grp] \\ &\sigma[URL.url = \texttt{www.cnn.com/health} \wedge Time.month \leq 1999/12](O)) \end{aligned} \tag{15}$$

As seen in Figure 1, $fact\_1$ satisfies both predicates at $2000/11/5$, making the predicates overlap. However, the ordering is not well-defined. The resulting problem is that if a_2 is effective, the predicate of a_3 cannot be evaluated, since information on $URL.url$ will no longer be available (a similar argument applies in the other direction). □

The next example illustrates how the $NonCrossing$ property enables supporting non-linear hierarchies. The potential problem occurs when two actions aggregate into parallel branches in the same dimension. Then, if there at some time will be an overlap in the facts reduced by these actions, it will be undefined which one is the largest and thus is the resulting granularity for those facts.

EXAMPLE: Assuming the two actions a_2 (equation 5) and a_4 (equation 16, below), we have actions aggregating into each of the branches in the non-linear hierarchy of the $Time$-dimension.

$$\begin{aligned} a_4 = \rho(\alpha[&Time.week, URL.url] \\ &\sigma[URL.url = \texttt{www.cnn.com/health} \wedge Time.month \leq 1999/12](O)) \end{aligned} \tag{16}$$

The predicates are identical to those in the previous example, so there is an overlap. Also the two actions are unordered. The consequence is that it cannot be determined which action aggregates to the highest granularity and therefore which one should be effective. □

We note that in a $NonCrossing$ specification, a unique *lowest available category type* always exists for any set of overlapping actions. Also, since we require that all facts initially have the bottom granularity, any action a_{init} will initially aggregate to a granularity $(\mathcal{C}_{1j_1}, \ldots, \mathcal{C}_{nj_n})$, such that the ordering $\leq_\mathcal{D}$ between this granularity and the granularity of the current fact f is well-defined.

Next, the $Growing$-property is necessary because actions may use the NOW-variable in their predicates. This variable may render actions either fixed, growing, or shrinking, and as we will illustrate, shrinking actions violate the irreversibility of data reduction.

Fixed - If the predicate refers only to fixed time values, the set of facts selected by an action does not vary across time.

Growing - If the predicate uses the NOW-variable *only* to set a growing upper bound and/or a decreasing lower bound for time values the set of facts selected by an action will grow as time passes.

Shrinking - If the predicate uses the NOW-variable to specify a *decreasing upper bound* and/or an *increasing lower bound* for time values, the set of facts selected by an action will be shrinking as time increases.

As data reduction is irreversible, when an action predicate shrinks and no longer specifies aggregation of a fact f, then immediately another action must specify at least the same level of aggregation for f. To be able to precisely ensure this, we define the $Growing$ property as follows.

$$\begin{aligned} Growing(V,O) \quad \stackrel{def}{\iff} \quad & \forall (v_1,\ldots,v_n) \in \bot_{D_1} \times \cdots \times \bot_{D_n} \; (\forall t_1, t_2 \; (t_1 < t_2 \wedge \\ & \forall i \in 1,\ldots,n \; (AggLevel_i(v_1,\ldots,v_n,t_1) \leq_{\mathcal{T}_i} AggLevel_i(v_1,\ldots,v_n,t_2)))) \end{aligned} \tag{17}$$

The definition states that if a cell $(v_1,\ldots,v_n)$ at time t_1 is reduced in the i'th dimension to an aggregation level, $AggLevel_i(v_1,\ldots,v_n,t_1)$, then at any later time t_2, this particular cell must be aggregated to at least the same level of aggregation in this dimension. This may be caused by any action within the data reduction specification.

EXAMPLE: Building on the MO presented in Figure 1 assume the data reduction specification $(\{a_1\}, \leq_{\mathcal{V}})$ containing only the single action specification a_1 (equation 4), which aggregates all facts referencing the `.com` $URL.domain\_grp$-value older than 6 months and younger than 12 months to the granularity of $(Time.month, URL.domain)$. Assuming the current time is month $2000/10$, all `.com`-facts referring to times in the interval $[1999/11; 2000/4]$ will be aggregated to the level of $Time.month$ and $URL.domain$, i.e., $fact\_0, fact\_12, fact\_3$, and $fact\_45$. Figure 2 assumes this and shows the situation as of time $2000/10$ in the upper-most set of facts (the upper most round-cornered box.) With this starting point, Figure 2 illustrates one situation to the left that violates the $Growing$-property, as well as a valid situation at the bottom. (In the figure, incomplete lines illustrate references that have not changed.) In the first situation, a violation is caused as follows. When a month has passed and the current time is $2000/11$, only facts referring to times in the interval $[1999/12; 2000/5]$ will be aggregated, meaning that $fact\_0$ will be reclaimed to the granularity of $(Time.day, URL.url)$. Such a reclaiming of $fact\_0$ is not possible and violates the $Growing$ property. One solution is to add action a_2 (equation 5).

This action aggregates all facts no longer captured by a_1 to $(Time.quarter, URL.domain)$. In the figure, the consequence of adding this action is shown in the lower-most situation. Here, $fact\_0$ and $fact\_3$ are aggregated to $fact\_03$, and $fact\_12$ is aggregated to a higher level. □

Together, these two properties ensure consistency and irreversibility when reducing a multidimensional object.

4.4 Semantics

We are now able to define the reduction of a multidimensional object.

DEFINITION 2 *Assume the multidimensional object $O = (\mathcal{S}, F, D, R, M)$, and the data reduction specification $V = (A, \leq_{\mathcal{V}})$ where $A = \{a_1,\ldots,a_k\}$, $Growing(V,O)$, and $NonCrossing(V)$. Then at time t, the reduced multidimensional object $O'(t) = (\mathcal{S}', F', D', R', M')$ is defined as follows. (Note that the set of measure values $M_j(f)$ is a multi-set.)*

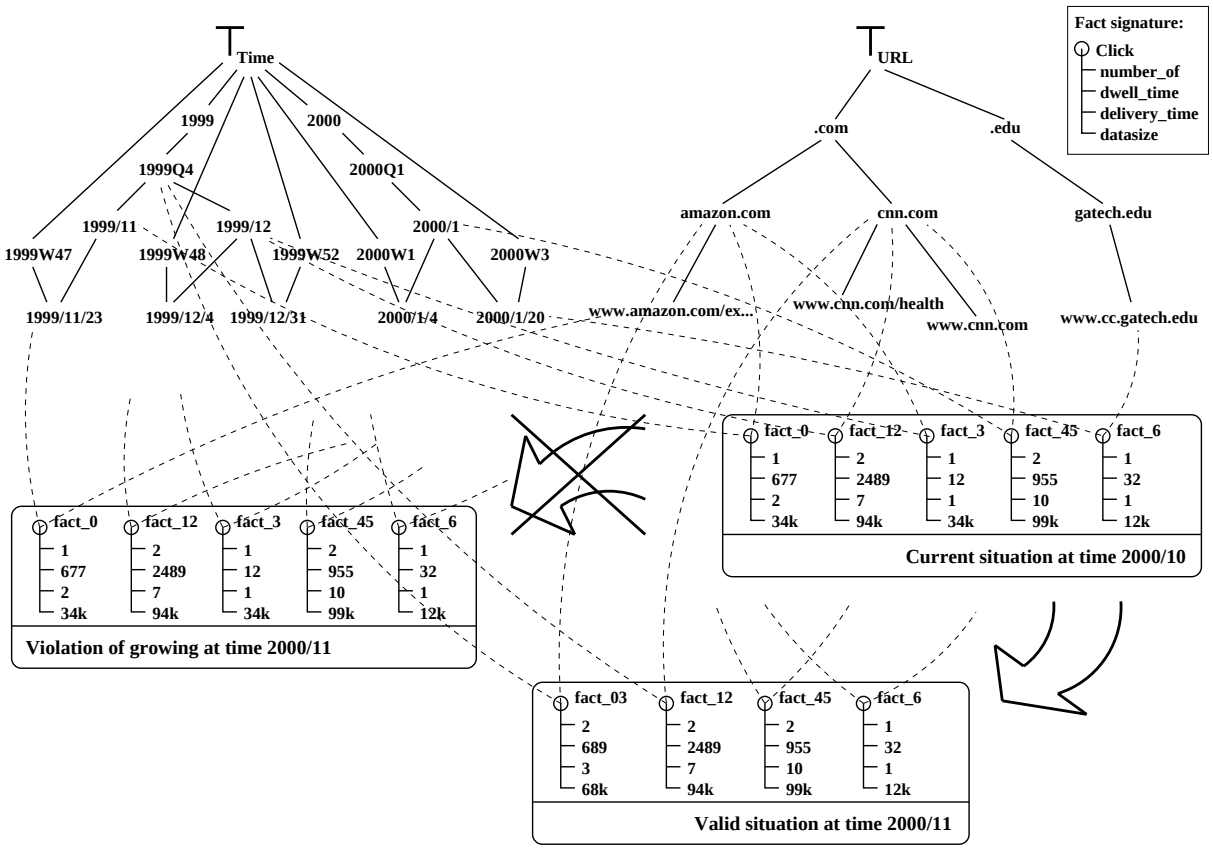

Figure 2: A $Growing$-Violating Situation and a Valid Data Reduction Situation

$$
\begin{array}{rcl}
\mathcal{S}' & \stackrel{def}{=} & \mathcal{S} \\
F' & \stackrel{def}{=} & \{f' \mid \exists (v'_1, \ldots, v'_n) \in D_1 \times \cdots \times D_n \; (f' = \{f \in F \mid Cell(f,t) = (v'_1, \ldots, v'_n)\})\} \\
D' & \stackrel{def}{=} & D \\
R' & \stackrel{def}{=} & \{R'_i \mid i = 1, \ldots, n\} \textit{ where } R'_i = \{(f', v'_i) \mid \forall f \in f' \; (Cell(f,t) = (v'_1, \ldots, v'_n))\} \\
M' & \stackrel{def}{=} & \{M'_j \mid j = 1, \ldots, m\} \textit{ where } M'_j(f') = a_{M_j}(\{M_j(f) \mid f \in f'\}) \textit{ and } a_{M'_j} = a_{M_j}
\end{array}
\tag{18}
$$

□

The definition states that the reduced object has the same schema and dimensions as the original object. Not changing the schema ensures that new facts conforming to the original schema may be inserted. (As an aside, it is possible to physically remove bottom-level category types if there is no use for them. Then the values of these category types will be discarded when facts are subsequently inserted.)

The reduced fact set F' is a set of facts f' where each one is a set of the original facts f having identical dimension values for all dimensions in the cell they are aggregated to. For each dimension, the relation set R'_i contains a relation for each fact to the dimension value of the category type (in this dimension) giving the granularity of the fact. Finally, the new set of measures has for each measure the value given by the default aggregation function when evaluated on the set of measure values on the facts aggregated to one new fact f'.

EXAMPLE: Again assume the MO from the ongoing example in Figure 1 and the data reduction specification $\{a_1, a_2\}$ (equations (4) and (5)). Then Figure 3 illustrates three snapshots of the reduced MO taken at three different time points. The first snapshot, at time $2000/4/5$, shows how none of the facts satisfy

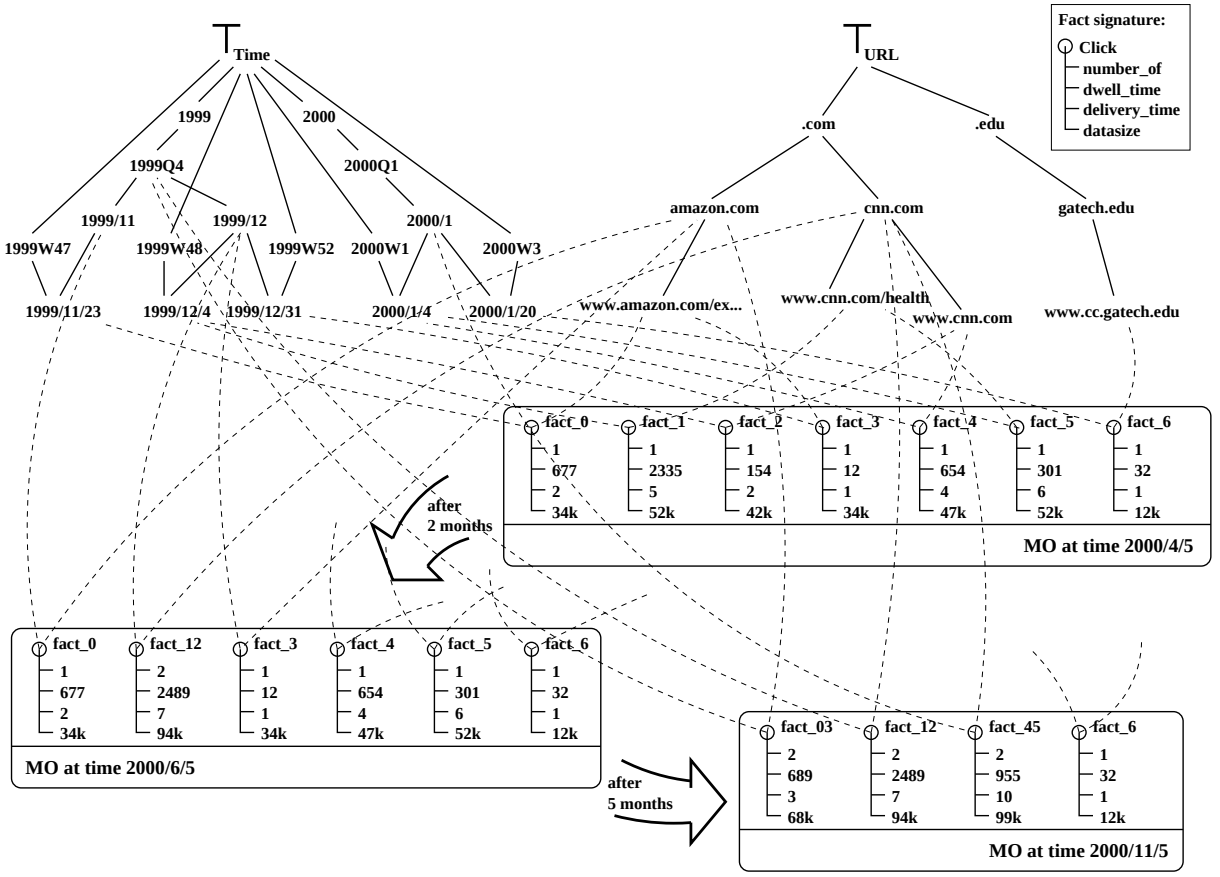

Figure 3: Three Snapshots of the MO Reduced by Data Reduction Specification $(\{a_1, a_2\}, \leq_{\mathcal{V}})$

the predicates; thus, no facts are aggregated. In the second snapshot, at time $2000/6/5$, all facts referring to time values in 1999 satisfy the predicate of a_1, so $fact\_1$ and $fact\_2$ are aggregated into $fact\_12$, with granularity $(Time.month, URL.domain)$, as well as both $fact\_0$ and $fact\_3$. (Facts with incomplete reference lines have not changed.) Finally, the third snapshot illustrates the situation after 5 more months, at time $2000/11/5$. We notice that $fact\_0$ and $fact\_3$ aggregate to one new fact and that $fact\_4$ and $fact\_5$ aggregate to another. At this time, all data referring 1999 have granularity $(Time.quarter, URL.domain)$. □

5 The Dynamics of Data Reduction

We proceed to consider the update of data reduction specifications. First, the necessary operators are defined, then the operational evaluation is considered.

5.1 Insertion and Deletion

With insertion and deletion operators for actions available, we are able to modify the data reduction as requirements or needs change. The $insert$-operator involves testing future consistencies such that the resulting specification stays $Growing$ and $NonCrossing$. Since these properties depend on the combination of actions and not on single action properties alone, a set of actions can only be inserted if the consistency is retained after inserting the full action set.

DEFINITION 3 *Let $A' = \{a'_1, \ldots, a'_m\}$ be a set of new actions, and let $V = (A, \leq_{\mathcal{V}})$ be the existing data reduction specification. Then the $insert$-operator is defined as follows.*

$$insert(A', V) \stackrel{def}{=} \begin{cases} V' & \text{if } Growing(V') \wedge NonCrossing(V') \quad \text{where } V' = (A \cup A', \leq_{\mathcal{V}}) \\ V & \text{otherwise} \end{cases} \tag{19}$$

□

The definition states that the updated action set must be $Growing$ and $NonCrossing$. If these are not satisfied, the original data reduction specification is left unchanged.

Inserting actions and checking the necessary consistency must be dependent on the action specifications only—the specific facts present in the MO should not affect the decision of whether or not a set of actions can be inserted. How this is satisfied is explained in the following two subsections.

Considering next the $delete$-operator, the wish is to be able to delete a set of actions if they have no effect on the MO, i.e., if they are not responsible for any aggregations. (An action is responsible for aggregation if its predicate is satisfied and no other action exists that aggregates the same fact(s) to a higher granularity.) Opposite to the $insert$-operator, the $delete$-operator must depend on the specific facts in the MO. The reason is as follows. If the responsibility was tested on all possible instances of the MO, there would for a reasonable action (actions with predicates that are never satisfied are uninteresting) exist an instance such that its predicate would evaluate to true for some facts. The action would thus be considered responsible for aggregations unless other actions exist to aggregate the same set of cells to higher granularities. As a result, very few actions may be deleted from the specification.

Consider a situation were no facts and actions exist and an action is inserted that is subsequently considered too radical. We then wish to delete this action, instead inserting a less radical action. If we were to consider all possible instances of the MO, we would not be able to perform the deletion, which is clearly unsatisfactory. Instead, we check the action against the facts actually in the MO, and the action can be deleted.

DEFINITION 4 *Let $A' = \{a'_1, \ldots, a'_m\}$ be a subset of the actions in the data reduction specification $V = (A, \leq_{\mathcal{V}})$. Then the $delete$-operator is defined as follows at time t.*

$$delete(A', V) \stackrel{def}{=}$$

$$\begin{cases} V' & \text{if } Growing(V') \wedge NonCrossing(V') \wedge \\ & \quad \forall a'_j \in A' \, (\forall f \in F \, (\exists (v_1, \ldots, v_n) \in D_1 \times \cdots \times D_n \, (\\ & \qquad [\forall i \in 1, \ldots, n \, ((f, v_i) \in R_i) \wedge (v_1, \ldots, v_n) \in Pred(a'_j, t)] \Rightarrow \\ & \qquad [Cat(a'_j) <_{\mathcal{D}} Gran(f) \vee (\exists a'' \in A \setminus A' \, ((v_1, \ldots, v_n) \in Pred(a'', t) \wedge \\ & \qquad\quad Cat(a'') =_{\mathcal{D}} Cat(a'_j)))] \,))) \\ & \text{where } V' = (A \setminus A', \leq_{\mathcal{V}}) \\ V & \text{otherwise} \end{cases} \tag{20}$$

□

The definition states that, in addition to maintaining the $Growing$ and $NonCrossing$-properties for the updated specification, the deleted actions should have no current effect on the MO. Thus, for all actions one more property is required for all facts in the MO. If the cell $(v_1, \ldots, v_n)$, to which the fact maps directly, is an element in the set of possible cells satisfying the action predicate, i.e., in $Pred(a'_j, t)$, then the granularity of the fact must be different from the granularity to which the action specifies aggregation. If this is satisfied, even though the fact satisfies the action predicate, the action is not responsible for the granularity (and aggregation) of the fact, and thus the action can be deleted. Note also that the operator follows the order-independent strategy of deleting all or no actions.

EXAMPLE: To illustrate the handling of NOW-relative actions, assume an action set containing only the action a_7 and the MO from the ongoing example.

$$a_7 = \rho(\ \alpha[Time.month, URL.domain]\ \sigma[Time.month \leq NOW - 12months](O)\ ) \tag{21}$$

Then to stop the action from having an effect on facts with times that are after year 1999, we observe that, in month $2000/12$, the action reduces all facts characterized by the month $1999/12$ or earlier. Therefore, a_7 can be deleted and thus stopped after insertion of action a_8.

$$a_8 = \rho(\ \alpha[Time.month, URL.domain]\ \sigma[Time.month \leq 1999/12](O)\ ) \tag{22}$$

If the two operations are performed during the month of $2000/12$, a_8 will aggregate the exact same facts as a_7, and since the aggregation level specified by the two actions are identical, $\{a_7, a_8\}$ is $Growing$ and $NonCrossing$. Also, after deleting a_7 the properties are maintained. Put differently, for all facts mapping to values (v_{Time}, v_{URL}), $(v_{Time}, v_{URL}) \in Pred(a_7, 1999/12) \Rightarrow ((v_{Time}, v_{URL}) \in Pred(a_8, 1999/12) \wedge Cat(a_8) =_{\mathcal{D}} Cat(a_7))$. □

5.2 Checking The *Non-Crossing* Property

We now consider how to check the $NonCrossing$-property in an *operational* way, i.e., how to check it in practice. Recall that a set of actions A was $NonCrossing$ if $\forall a_1, a_2 \in A(\exists t \in T\ (P_1(t) \wedge P_2(t) \Rightarrow a_1 \leq_V a_2 \vee a_2 \leq_V a_1))$, where $P_1(t)$ is short for $Pred(a_1, t)$ (recall Equation (14)). The algorithm below provides an operational way of checking whether the non-crossing property holds for a given pair of actions. Using this, checking whether a set A is non-crossing requires $\mid A \mid^2$ calls to the algorithm. Given that the number of actions in a specification is relatively low and that this checking only is necessary when the set of actions is updated, this algorithm offers ample performance.

```
1)    FUNCTION noncrossing (a1, a2)
2)       IF a1 ≤V a2 ∨ a2 ≤V a1 THEN RETURN true;
3)       IF P1 and P2 are independent of time THEN RETURN ¬(P1 ∧ P2);
4)       IF ∃t ∈ T (P1(t) ∧ P2(t)) THEN RETURN false;
5)       RETURN true;
```

The algorithm works as follows. If the two actions a_1 and a_2 are ordered with respect to each other (line 2), which is easily checked syntactically, the property is true. Otherwise, if their predicates P_1 and P_2 are independent of time (again, an easy syntactic check), the formula contains no variables, and $P_1 \wedge P_2$ can easily be checked using a standard theorem prover such as PVS [12]. In the final case, we need to check whether the predicates are simultaneously true at any time (line 4), which may again be proved using a standard theorem prover.

5.3 Checking The *Growing* Property

While the definition of the $Growing$ property in equation (17) is intuitive, it is not operational because it refers to all possible MOs. Thus, we need to come up with *operational* rules for evaluating the $Growing$-property. These rules should allow us to determine whether the property will be maintained for the whole system, e.g., after inserting a given action. It should be possible to determine this solely on the basis of the given set of actions. When describing the operational rules in the following, we distinguish between two distinct situations. In the first case we provide a proof that this by default creates a $Growing$ specification, and in the second case we provide an algorithmic solution for determining if the result will be $Growing$ and we argue that the algorithm provides the correct answer. First, however, we need to outline a common *pre-processing* step for the two distinct situations, in order to ease the check of the $Growing$-property.

As mentioned above, we insert *a set* of actions at the same time. The *pre-processing step* first transforms the predicates in the given set of actions into *disjunctive normal form* [3]. Second, it *splits* each action into a number of actions, one for each disjunctive part of the predicate, e.g., the action "$\alpha[C_1, \ldots, C_n]\sigma[A \vee B](O)$" is split into the two actions "$\alpha[C_1, \ldots, C_n]\sigma[A](O)$" and "$\alpha[C_1, \ldots, C_n]\sigma[B](O)$." This normalized set has exactly the same effect as the old set, and the predicate of any action is a conjunction of (range) predicates on each of the dimensions, i.e., the predicate can be evaluated by evaluating a (range) predicate on each dimension. For a dimension d with no predicate, the predicate is $d.\top = \top_d$ (always true), is assumed.

The growing property for a set of actions holds if and only if it holds for each action in the context of the set. Thus, when inserting a set of actions, we only need to check the property for each new action as it holds by assumption for all existing actions. When deleting a set of actions, we have to check whether the remaining actions still satisfy the property.

The first of the two cases occurs when an action is itself growing. An action is $Growing$ if any cell once satisfying the action predicate will continue to do so. In this case, the total set of actions will also be $Growing$ after the insertion. This is stated by the following theorem.

THEOREM 1 Let a be a growing action and $V = (A, \leq_{\mathcal{V}})$ be a data reduction specification that satisfies the $Growing$-property. Then the specification $V' = (A \cup \{a\}, \leq_{\mathcal{V}})$ is $Growing$. □

PROOF Let the current time be t and $(v_1, \ldots, v_n)$ be any cell. There are three cases:

1. $(v_1, \ldots, v_n)$ satisfies the predicate of a at time t, i.e., $(v_1, \ldots, v_n) \in Pred(a, t)$
2. $(v_1, \ldots, v_n)$ does not satisfy the predicate of a at time t, but will do so at a later time t_1, i.e., $(v_1, \ldots, v_n) \notin Pred(a, t) \wedge \exists t_1 > t\ ((v_1, \ldots, v_n) \in Pred(a, t_1))$
3. $(v_1, \ldots, v_n)$ will never satisfy the predicate of a, i.e., $\nexists t\ ((v_1, \ldots, v_n) \in Pred(a, t))$

In case 1), $(v_1, \ldots, v_n)$ will continue to satisfy a's predicate as a is growing. This means that $(v_1, \ldots, v_n)$ will remain to be aggregated to at least the level specified by a. If $(v_1, \ldots, v_n)$ is aggregated even higher by an action in A, growing will hold by the assumption on A. Thus, the growing property will always hold. In case 2), the reasoning is similar as $(v_1, \ldots, v_n)$ will continue to satisfy a's predicate once it has done so initially. In case 3), $(v_1, \ldots, v_n)$ will only be affected by the actions in A which by assumption satisfies the growing property. As $(v_1, \ldots, v_n)$ was any cell, the growing property is satisfied for any MO.

Q.E.D.

Growing actions include many of the most common types of actions, and they encompass actions with predicates that have (A) fixed boundaries, (B) no lower boundary and an increasing upper boundary, (C)

no upper boundary and a decreasing lower boundary, (D) a fixed lower boundary and an increasing upper boundary, and (E) a fixed upper boundary and a decreasing lower boundary.

As variables occur in predicates only for the time dimension, all predicates on non-time dimensions fall in category A, and the possibly moving boundaries always relate to the time dimension. Remembering that the predicates were reduced to range predicates on each dimension, we observe that the above rules can easily be syntactically checked for a given predicate.

EXAMPLE: The above cases cover the most likely predicates to be used in a real setting. For example, the set of rules *"aggregate to month after two years"* and *"aggregate to quarter after four years"* fall in category B. As a stronger action *dominates* a weaker one, it is not really necessary to have a lower boundary on the time dimension. □

The second case, where the action itself is not $Growing$, appears when the given action has one of the following types of predicates: (F) an increasing lower boundary, (G) a decreasing upper boundary, and (H) an increasing lower boundary and a decreasing upper boundary.

To maintain the property for the total set of actions together with an action a of one of these types requires a set of actions that maintains the aggregation level for all facts once aggregated by the given action. We proceed to present an algorithmic approach for evaluating if this is satisfied. We only show the solution for category F. The solution for category G is similar, just reversed, and category H can be handled by combining the solutions for F and G.

We check the $Growing$ property in three steps. In the first, we create the time function t_{N_a} that specifies the variable lower boundary (on the Time dimension) of a. Due to the format of the predicates (conjunctions), we always have one unique lower boundary in each dimension.

In the second step, we find the action set $A' = \{a_j \mid a \leq_V a_j\}$ of actions that can possibly help to make action a "safe," by "catching" the cells that over time "fall over" the (increasing) lower boundary of a. As the lower boundary of a is given by t_{N_a}, the first time point where a cell will no longer satisfy the predicate of a is $t_{N_a} - 1$, where "1" is one unit in the finest time granularity. Thus, there must exist one or more actions in A' that can "take over" from a at time $t_{N_a} - 1$.

In the third step, we perform the predicate check

$$P[Time.\texttt{<category>} \leftarrow t_{N_a}] \Rightarrow \bigvee_j P_j[Time.\texttt{<category>} \leftarrow (t_{N_a} - 1)], \tag{23}$$

where P is the predicate of a and the P_j's are the predicates of the a_j's. Here, the reference to the time dimension in each predicate, $Time.\texttt{<category>}$, is replaced by the lower boundary (t_{N_a}) on the left side and the lower boundary minus one time unit on the right side. Due to the specification of t_{N_a}, the time part of the predicate on the left side is always true and can thus be removed from the conjunction. This predicate check can be done by a standard theorem prover as PVS [12].

This three-step algorithmic approach checks the maintenance of the $Growing$-property in the distinct situation where the action itself is not growing. This is also illustrated by the following example.

EXAMPLE: Let the set of actions (in a two-dimensional case from the ongoing example) be as follows.

$$\begin{aligned} a_1 = \alpha[&Time.month, URL.domain] \\ &\sigma[NOW - 4years \leq Time.year \leq NOW \wedge URL.\top = \top_{URL}](O) \end{aligned} \tag{24}$$

$$\begin{aligned} a_2 = \alpha[&Time.quarter, URL.domain] \\ &\sigma[Time.year < NOW - 4years \wedge URL.domain\_grp = \texttt{.com}](O) \end{aligned} \tag{25}$$

$$\begin{aligned} a_3 = \alpha[&Time.quarter, URL.domain\_grp] \\ &\sigma[Time.year < NOW - 4years \wedge URL.domain\_grp = \texttt{.edu}](O) \end{aligned} \tag{26}$$

Thus, we aggregate data to $month$ and $domain$ until facts are four years old, then we aggregate facts to $quarter$ and $domain$ for `.com` URL-values and to $quarter$ and $domain\_grp$ for `.edu` URL-values. We now need to check whether this set is growing. The rules a_2 and a_3 are growing in themselves (case B), so we need only check rule a_1. The lower bound is given by $t_{N_{a_1}} = NOW - 4years$. We find that $A' = \{a_2, a_3\}$. We now need to check the following.

$$\begin{aligned} &P_1[Time.year \leftarrow NOW - 4years] \Rightarrow \\ &\quad P_2[Time.year \leftarrow NOW - 4years - 1day] \ \vee \ P_3[Time.year \leftarrow NOW - 4years - 1day] \end{aligned} \tag{27}$$

This is expanded to

$$\begin{aligned} &NOW - 4years \leq NOW - 4years \leq NOW \wedge URL.\top = \top_{URL} \Rightarrow \\ &\quad (NOW - 4years - 1day < NOW - 4years \wedge URL.domain\_grp = \texttt{.com}) \vee \\ &\quad (NOW - 4years - 1day < NOW - 4years \wedge URL.domain\_grp = \texttt{.edu}) \end{aligned} \tag{28}$$

A theorem prover capable of simple formula rewriting [12] and the rule about the time part on the left side can reduce this to

$$URL.\top = \top_{URL} \Rightarrow URL.domain\_grp = \texttt{.com} \ \vee \ URL.domain\_grp = \texttt{.edu} \tag{29}$$

as all the predicates on time evaluate to true. This can easily be proved by a theorem prover with knowledge of the domain of the URL dimension. □

6 Query Language

This section defines an algebraic query language that precisely describes the semantics of queries on reduced MOs. The language is not computationally complete, but includes only the operators corresponding to standard OLAP functions, i.e., selection, projection, and aggregation. Additional operators are described elsewhere [13]. These operators are left out, purposefully, to make sure that the computational power of the language will not surpass that of any commercial OLAP tool, rendering the results presented here widely applicable to commercial OLAP tools.

A reduced MO has a set of facts with varying granularities. Specifically, two problems need to be addressed, one concerning the selection operator, and one concerning the aggregate formation operator, both related to handling dimension values of varying granularities.

6.1 The Selection Operator

The varying-granularity problem in relation to selection is that the data might not be precise enough to determine whether the selection predicate holds: the category on which the query predicate is based might not be available because the data is aggregated to a higher category. For example, if the predicate concerns days and the data is aggregated to months, great care has to be taken to determine which facts should be selected.

Generally, three approaches can be taken to handling this problem [13]. The *conservative* approach returns only the facts that are *known* to satisfy the predicate, i.e., where the dimension values are totally within the range asked for. The *liberal* approach returns all the facts that *might* satisfy the predicate, i.e., where the dimension values at least overlaps the range asked for. Finally, the *weighted* approach returns facts that might satisfy the predicate, but attaches a weight to the fact based on the certainty of predicate satisfaction. Which approach to choose depends on the application. In this paper, we assume the *conservative* approach

since we believe this to be the most useful for data warehouse applications. The other approaches may be handled similarly [13].

EXAMPLE: Assume the $Time$-dimension and the reduced MO presented in the ongoing example, based on the data reduction specification $(\{a_1, a_2\}, \leq_\nu)$ at time $2000/11/5$ (see Figure 3).

The $Time$-dimension has a parallel hierarchy of category types, and therefore the following queries illustrate some interesting selection problems.

$$Q_1 = \sigma[Time.quarter \leq 1999Q3](O) \tag{30}$$
$$Q_2 = \sigma[Time.month \leq 1999/10](O) \tag{31}$$
$$Q_3 = \sigma[Time.week < 1999W48](O) \tag{32}$$

Query Q_1 is simple and unaffected by data reduction since all data has granularities lower than or equal to $Time.quarter$. Query Q_2 selects on month $1999/10$, and since the facts referring year 1999 has granularity $Time.quarter$, the facts mapping to dimension value $1999Q4$ satisfies the predicate only partly. Since we use the *conservative* approach those facts will not be included in the answer. Finally, query Q_3 selects on week $1999W48$. Since weeks are not directly related to quarters through the partial ordering between categories, when evaluating the facts referring to $Time.quarter$ values it is necessary to look at the $Time.day$-category which is a category that both categories are directly related to through the ordering. So, this can be evaluated by comparing the days included in weeks and months, respectively. □

To ensure this behavior when comparing dimension values of two different granularities, we *drill down* the values to a common, possibly lower, granularity and compare the resulting values at this common granularity (as illustrated above). For example, when comparing month values with day values, we can drill down a month value to its corresponding day values using the dimension hierarchy and compare the resulting day values to the day values in the predicate. This is non-trivial due to the semantics of the different comparison operators. Specifically, the non-reflexive inequalities $(<, >)$, the reflexive inequalities $(\leq, \geq)$, the equalities $(=, \neq)$, and the inclusion $(\in)$ needs to be handled differently.

To find a common granularity, and thus to *drill down* values, we define an auxiliary function $GLB_i : 2^{C_i} \mapsto C_i$. Assuming that the i'th dimension forms a lattice GLB_i takes a number of categories and returns the single category in this dimension that is the *greatest lower bound* of these input categories, i.e., of all categories of lower granularity than all input categories, the return category has the highest granularity. For any set of categories in the i'th dimension, $(C_{i1}, \ldots, C_{ik})$, GLB_i is defined as follows.

$$\begin{aligned} GLB_i(C_{i1}, \ldots, C_{ik}) \stackrel{def}{=} C_i' \quad \text{where } & (C_i' \leq_{\mathcal{T}_i} C_{i1} \wedge \cdots \wedge C_i' \leq_{\mathcal{T}_i} C_{ik}) \wedge \forall C_{ij} \in C_i \, (\\ & (C_{ij} \leq_{\mathcal{T}_i} C_{i1} \wedge \cdots \wedge C_{ij} \leq_{\mathcal{T}_i} C_{ik}) \Rightarrow C_{ij} \leq_{\mathcal{T}_i} C_i' \,) \end{aligned} \tag{33}$$

Note, that when the dimension categories do not form a lattice, we still know that there is *at least one* lower bound for any two categories C_{ij} and C_{ik}. This is due to the bottom category $\perp_i$. In this case, it will not be possible to select the lower bound specifying the highest granularity. This does not pose a problem since any lower bound will do, so below we can assume a lattice and thus a *unique* greatest lower bound.

EXAMPLE: For the $Time$-dimension in our example $GLB_{Time}(Time.week, Time.quarter) = Time.day$ since $Time.day \leq_{Time} Time.week \wedge Time.day \leq_{Time} Time.quarter$ and since no other larger category exists that satisfies this property. □

Using this auxiliary function the semantics of the comparison operators are defined as follows.

DEFINITION 5 *Let f be a fact in F, and let v' be a dimension value to which f maps directly, i.e., if v' is a value in the i'th dimension $D_i = (C_i, \leq_{D_i})$ then $(f, v') \in R_i$. Let v' belong to the category C'_i. Also, let $v_1, v_2, \ldots, v_k$ be values in the domain of D_i, and let these values belong to the categories $C_{i1}, C_{i2}, \ldots, C_{ik}$, respectively. Then the operators are defined as follows.*

$$v' \textbf{ op } v_1 \iff \begin{cases} \forall v_a \in GLB_i(C'_i, C_{i1}) \left(\forall v_b \in GLB_i(C'_i, C_{i1}) \left(\right.\right. \\ \qquad (v_a \leq_{D_i} v' \wedge v_b \leq_{D_i} v_1) \Rightarrow v_a \textbf{ op } v_b\left.\right)\right) & \text{if op } \in \{<, >\} \\ \forall v_a \in GLB_i(C'_i, C_{i1}) \left(\exists v_b \in GLB_i(C'_i, C_{i1}) \left(\right.\right. \\ \qquad (v_a \leq_{D_i} v' \wedge v_b \leq_{D_i} v_1) \Rightarrow v_a \textbf{ op } v_b\left.\right)\right) & \text{if op } \in \{\leq, \geq\} \\ \{v_a \mid v_a \in GLB_i(C'_i, C_{i1}) \wedge v_a \leq_{D_i} v'\} \textbf{ op} \\ \qquad \{v_b \mid v_b \in GLB_i(C'_i, C_{i1}) \wedge v_b \leq_{D_i} v_1\} & \text{if op } \in \{=, \neq\} \end{cases} \tag{34}$$

$$\begin{aligned} v' \in \{v_1, \ldots, v_k\} \iff\ & \forall v_a \in GLB_i(C'_i, C_{i1}, C_{i2}, \ldots, C_{ik}) \left(\right. \\ & \exists v_b \in GLB_i(C'_i, C_{i1}, C_{i2}, \ldots, C_{ik}) \left((v_a \leq_{D_i} v' \wedge \exists j \in 1, \ldots, k\ (v_b \leq_{D_i} v_j)) \Rightarrow v_a = v_b\right)\left.\right) \end{aligned} \tag{35}$$

□

All operators use GLB_i which, assuming a lattice, gives the greatest lower bound which is exactly the common category of highest granularity, on which the values are successfully compared. Note, if $Type(v') \geq_{\mathcal{T}_i} Type(v_1)$ then GLB_i will return the category of v_1 and there will only be one possible v_b satisfying $v_b \leq_{D_i} v_1$, similarly if $Type(v') \leq_{\mathcal{T}_i} Type(v_1)$.

For the strong operators $<$ and $>$ (see Eq. 34) the definition states that to compare one value v_1 to another v_2, all drill-down values created from v_1 must compare the same way to all drill-down values created from v_2.

EXAMPLE: Based on the example assume $fact\_03$ and query Q_3, then the expression $P[fact\_03]$ in question is $1999Q4 < 1999W48$. Then both values are drilled down to the greatest lower bound, i.e., days. Then the definition states that for all days (in the $Time$-dimension) from $1999Q4$, i.e., $\forall v_a \in Time.day(v_a \leq_{Time} 1999Q4)$, if all days in week $1999W48$ are larger, then the expression evaluates to $TRUE$. So in the example we evaluate the following.

$$\begin{aligned} &\forall v_1 \in \{1999/12/4\}\ (1999/11/23 < v_1) \wedge \\ &\forall v_2 \in \{1999/12/4\}\ (1999/12/4 < v_2) \wedge \\ &\forall v_3 \in \{1999/12/4\}\ (1999/12/31 < v_3) \end{aligned}$$

Since $1999/12/31 \not< 1999/12/4$ this example evaluates to $FALSE$. However had the expression been $1999Q4 < 2000W1$, then the expression would evaluate to $TRUE$.

Note, since the example does not include all detail values in its dimensions, week $1999W48$ consists of only one day, as quarter $1999Q4$ consists of only 3 days. □

For the weaker operators $\leq$ and $\geq$ (see Eq. 34,) the definition states in a similar way that to compare one value to another, all the drill-down values created from the one must compare the same way to at least one drill-down value created from the other.

The equality and in-equality operators are slightly different (see Eq. (34),) since equality between two values requires the sets of drill-down values created from either of those two values must be identical. In practice this will usually mean, that equality is only possible when comparing values from the same category.

Finally, the $\in$-operator (see Eq. 35) compares one value with a set of values. The definition states that if for all drill-down values created from the one value there exists a value equal to it drilled down from any value in the set to compare with, then the expression evaluates to $TRUE$.

EXAMPLE: Assume the expression $1999Q4 \in \{1999W39, \ldots, 2000W1\}$. Again, the greatest lower bound is the day-category. The value $1999Q4$ drills down to $\{1999/11/23, 1999/12/4, 1999/12/31\}$ and the set $\{1999W39, \ldots 2000W1\}$ drills down to $\{1999/11/23, 1999/12/4, 1999/12/31, 2000/1/4\}$, we need to evaluate the following.

$$\begin{aligned}
&\exists v_1 \in \{1999/11/23, 1999/12/4, 1999/12/31, 2000/1/4\}\ (1999/11/23 = v_1) \wedge \\
&\exists v_2 \in \{1999/11/23, 1999/12/4, 1999/12/31, 2000/1/4\}\ (1999/12/4 = v_2) \wedge \\
&\exists v_3 \in \{1999/11/23, 1999/12/4, 1999/12/31, 2000/1/4\}\ (1999/12/31 = v_3)
\end{aligned}$$

This evaluates to $TRUE$. Similarly, the expression $1999Q4 \in \{1999W39, \ldots, 1999W51\}$ evaluates to $FALSE$. □

Having defined the operators when trying to compare values from both identical and different categories within the same dimension, the *conservative* selection-approach is handled. The *liberal* and *weighted* approaches can be handled in similar manners.

Given a predicate p on the dimension types $\mathcal{D} = \{\mathcal{T}_i\}$, we define the selection operator, σ, as:

$$\sigma[p](O) \stackrel{def}{=} (\mathcal{S}', F', D', R', M'), \tag{36}$$

where $\mathcal{S}' = \mathcal{S}$, $F' = \{f \in F \mid \exists v_1 \in D_1, \ldots, v_n \in D_n\ (\ p(v_1, \ldots, v_n) \wedge f \rightsquigarrow_1 v_1 \wedge \cdots \wedge f \rightsquigarrow_n v_n)\}$, $D' = D$, $R' = \{R'_i\}$, $R'_i = \{(f', v_i) \in R_i \mid f' \in F'\}$, and $M' = M|_{F'}$. Thus, we restrict the set of facts to those characterized by values where p evaluates to true. The fact-dimension relations and the measures are restricted accordingly, while the dimensions and the schema stay the same.

6.2 The Projection Operator

Defining the projection operator, we assume, without loss of generality, that the projection is over the k dimensions $D_1, \ldots, D_k$ and the l measures $M_1, \ldots, M_l$.

$$\pi[D_1, \ldots, D_k][M_1, \ldots, M_l](O) \stackrel{def}{=} (\mathcal{S}', F', D', R', M'), \tag{37}$$

where $\mathcal{S}' = (\mathcal{F}', \mathcal{D}', \mathcal{M}')$, $\mathcal{F}' = \mathcal{F}$, $\mathcal{D}' = \{\mathcal{T}_1, \ldots, \mathcal{T}_k\}, \mathcal{M}' = \{\mathcal{M}_1, \ldots, \mathcal{M}_l\}, F' = F$, $D' = \{D_1, \ldots, D_k\}$, $R' = \{R_1, \ldots, R_k\}$, and $M' = \{M_1, \ldots, M_l\}$, Thus, we retain only the k dimensions and the l measures, but the set of facts stays the same. Note that we do not remove "duplicate values." Thus the same combination of dimension values may be associated with several facts, as is the case in regular star schemas.

EXAMPLE: Assuming the query $\pi[URL][Number\_of, Dwell\_time](O)$ on the reduced MO from the running example at time $2000/11/5$, then the resulting MO is illustrated in Figure 4. □

6.3 The Aggregate Formation Operator

The aggregate formation operator α computes aggregate functions on the measures of an MO [9]. Given an n-dimensional MO O and a set of categories $C_i \in D_i$, $i \in 1, \ldots, n$, the query $\alpha[C_1, \ldots, C_n](O)$ aggregates the facts in O to the levels $C_1, \ldots, C_n$. Before formally defining the operator, we address the

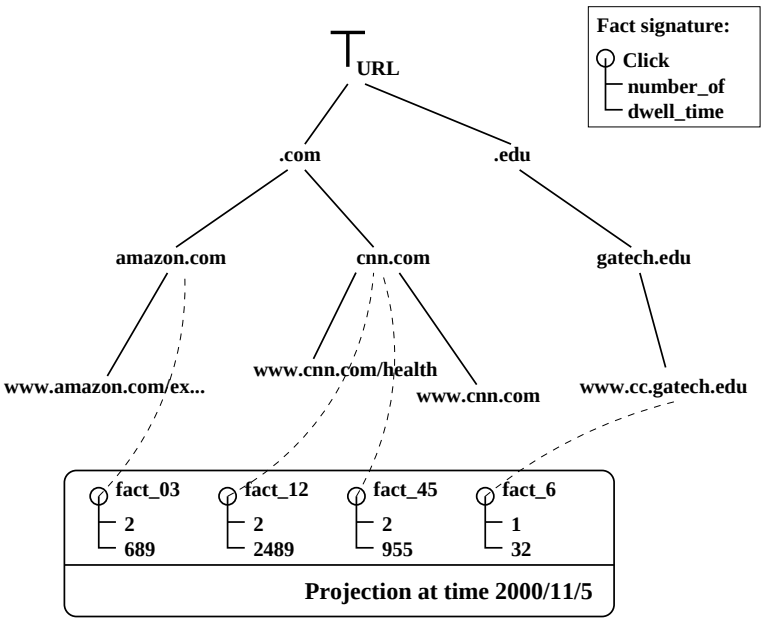

Figure 4: Result of Query $\pi[URL][Number\_of, Dwell\_time](O)$ at Time $2000/11/5$

varying-granularity problem, that the aggregation level asked for in the query might not be available for all facts. Assume a query that aggregates the data to the categories $C_{1j_1}, \ldots, C_{nj_n}$. If some argument facts have only dimension values in the i'th dimension of granularity $C_{ij_i'}$ where $C_{ij_i} <_{\mathcal{T}_i} C_{ij_i'}$, i.e., dimension values of a higher granularity than requested, the issue is whether such facts should be reflected in the result.

Four approaches may be envisioned. The *strict* approach considers only facts characterized by dimension values of the desired granularity, or lower. This guarantees that facts in the result has the correct, desired granularity. The *Least Upper Bound* (LUB) approach returns facts with the finest common granularity that is greater than or equal to the desired granularity and available for all of argument facts, i.e., the LUB of the desired granularity and the granularities of the input facts. The *availability* approach returns facts with a granularity corresponding to the finest available granularity for each fact that is greater than or equal to the desired one. The *disaggregated* approach returns facts with the granularity asked for, i.e., the granularity of C_{ij_i}. To achieve this *disaggregation* might have to be applied, yielding imprecise answers [5].

Which approach to take depends on the importance of reflecting all available facts in the answer and of obtaining an answer that has one single granularity or the correct granularity. In the sequel, we assume the availability approach because this gives the most detailed answer that is still guaranteed to be correct. The solutions for the strict and LUB approaches are special cases of the solution for the availability approach.

EXAMPLE: Assume the reduced MO at time $2000/11/5$ as presented in Figure 3. All facts referring time values in 1999 are aggregated to the granularity of $Time.quarter$, and all facts referring the url's in the domain group of `.com` are aggregated to the granularity of $URL.domain$. The queries $Q_4 = \alpha[Time.year, URL.domain]\ (O)$ and $Q_5 = \alpha[Time.month, URL.domain]\ (O)$ illustrate the aggregation problem explained above. These queries aggregate to $Time.year$ and $Time.month$, respectively. For Q_4, the full answer will have the specified granularity, since both $Time.year$ and $URL.domain$ are available for all facts. However, for Q_5, $(Time.month, URL.domain)$ is the desired aggregation level, and neither $fact\_03$ nor $fact\_12$ have values of lower granularity than $Time.quarter$. With the availability approach, they should be aggregated to this level in this dimension, but to the specified levels in all other dimensions. All other facts should be aggregated to the specified level of $Time.month$. □

To ensure this behavior we define an auxiliary function $Group\_high$ that groups facts characterized by the same set of dimension values above a certain granularity. Also, given a specified granularity it ensures that for those values of higher granularity in such a set, the facts are not only characterized by these values, but are mapped to them through relationships in R_i. Assume an n-dimensional MO, $O = (\mathcal{S}, F, D, R, M)$, a desired granularity level $(C_{1j_1}, \ldots, C_{nj_n})$ on O, and an n-tuple $(v_1, \ldots, v_n)$ where $\forall i \in 1, \ldots, n\ (Type(v_i) \geq_{\mathcal{T}_i} \mathcal{C}_{ij_i})$. Then we define $Group\_high$ as follows.

$$\begin{aligned} &Group\_high((v_1, \ldots, v_n), (C_{1j_1}, \ldots, C_{nj_n})) \stackrel{def}{=} \\ &\quad \{f \mid f \in F \wedge \forall i \in 1, \ldots, n\ (f \rightsquigarrow_i v_i \wedge (Type(v_i) >_{\mathcal{T}_i} \mathcal{C}_{ij_i} \Rightarrow (f, v_i) \in R_i))\} \end{aligned} \tag{38}$$

The function groups all facts characterized by all values in the n-tuple $(v_1, \ldots, v_n)$ and even mapped directly to those values v_i of a granularity higher than specified, i.e., $Type(v_i) >_{\mathcal{T}_i} \mathcal{C}_{ij_i}$. This ensures that all facts of a lower granularity in those dimensions D_i will not be in this group.

EXAMPLE: In the example consider the following based on the reduced MO at time $2000/11/5$.

$$\begin{aligned} Group\_high((1999Q4, \texttt{amazon.com}), (Time.month, URL.domain)) &= \{fact\_03\} \\ Group\_high((1999, \texttt{amazon.com}), (Time.month, URL.domain)) &= \emptyset \\ Group\_high((2000/1, \texttt{gatech.edu}), (Time.month, URL.domain)) &= \{fact\_6\} \end{aligned}$$

These examples show how facts of low granularity, here $fact\_6$, are aggregated to a higher level, while facts mapping to higher granularities are aggregated no further in that dimension, i.e., $fact\_03$ is aggregated to its own granularity $Time.quarter$, but not to $Time.year$. □

DEFINITION 6 *Assume an n-dimensional MO, $O = (\mathcal{S}, F, D, R, M)$ and a set of categories $\{C_{1j_1}, \ldots, C_{nj_n}\}$ where $\forall i \in 1, \ldots, n\ (C_{ij_i} \in D_i)$. Then the aggregate formation operator is defined as*

$$\alpha[C_{1j_1}, \ldots, C_{nj_n}](O) \stackrel{def}{=} (\mathcal{S}', F', D', R', M'), \tag{39}$$

where

$$\begin{aligned} &\mathcal{S}' = (\mathcal{F}', \mathcal{D}', \mathcal{M}'),\ \textit{where } \mathcal{F}' = 2^{\mathcal{F}}, \mathcal{D}' = \{\mathcal{T}_i' \mid i \in 1, \ldots, n\}, \mathcal{M}' = \mathcal{M}, \mathcal{T}_i' = (\mathcal{C}_i', \leq_{\mathcal{T}_i}', \perp_{\mathcal{T}_i}', \top_{\mathcal{T}_i}'), \\ &\quad \mathcal{C}_i' = \{\mathcal{C}_{ij} \in \mathcal{T}_i \mid Type(C_{ij_i}) \leq_{\mathcal{T}_i} \mathcal{C}_{ij}\}, \leq_{\mathcal{T}_i}' = \leq_{\mathcal{T}_i|_{\mathcal{C}_i'}}, \perp_{\mathcal{T}_i}' = Type(C_{ij_i}), \top_{\mathcal{T}_i}' = \top_{\mathcal{T}_i} \\ &F' = \{Group\_high((v_1, \ldots, v_n), (C_{1j_1}, \ldots, C_{nj_n})) \mid \exists C_{1j_1'} \times \cdots \times C_{nj_n'}\ (\\ &\qquad \forall i \in 1, \ldots, n\ (C_{ij_i'} \geq_{\mathcal{T}_i} C_{ij_i}) \wedge (v_1, \ldots, v_n) \in C_{1j_1'} \times \cdots \times C_{nj_n'}) \wedge \\ &\qquad Group\_high((v_1, \ldots, v_n), (C_{1j_1}, \ldots, C_{nj_n})) \neq \emptyset\} \\ &D' = \{D_i' \mid i \in 1, \ldots, n\},\ \textit{where} \\ &\quad D_i' = (C_i', \leq_{D_i}'), C_i' = \{C_{ij}' \in D_i \mid Type(C_{ij}') \in \mathcal{C}_i'\}, \leq_{D_i}' = \leq_{D_i|_{D_i'}} \\ &R' = \{R_i' \mid i \in 1, \ldots, n\},\ \textit{where} \\ &\quad R_i' = \{(f', v_i') \mid \exists C_{1j_1'} \times \cdots \times C_{nj_n'}\ (\\ &\qquad \forall i \in 1, \ldots, n\ (C_{ij_i'} \geq_{\mathcal{T}_i} C_{ij_i}) \wedge \exists (v_1, \ldots, v_n) \in C_{1j_1'} \times \cdots \times C_{nj_n'}\ (\\ &\qquad f' \in F' \wedge f' = Group\_high((v_1, \ldots, v_n), (C_{1j_1}, \ldots, C_{nj_n})) \wedge v_i = v_i'))\} \\ &M' = \{M_j' \mid j \in 1, \ldots, m\},\ \textit{where } M_j'(f') = a_{M_j}(\{M_j(f) \mid f \in f'\}), a_{M_j'} = a_{M_j} \end{aligned} \tag{40}$$

□

In the definition, restrictions on the fact set F' and fact-dimension relations R_i handle the varying-granularity issues. For F' the approach is to $Group\_high$ on the value combinations of exactly the wanted granularity and also on the value combinations of higher granularities. This way, the operator catches facts of higher granularities than the one wanted in the aggregation result. Doing this, the restriction of $Type(v_i) >_{\mathcal{T}_i} C_{ij_i} \Rightarrow (f, v_i) \in R_i$ in $Group\_high$ prevents facts from being aggregated to more than one granularity and thus into more than one new fact. A similar restriction is put on R_i'.

EXAMPLE: Again considering the ongoing example, the answer to query Q_5, evaluated at time $2000/11/5$, is given in Figure 5. As the illustration of the $Group\_high$ function in the example above, $fact\_45$ and $fact\_6$ both aggregate to $Time.month$, while $fact\_03$ and $fact\_12$ both are aggregated to granularity $Time.quarter$ since no lower granularity is available for the two facts. □

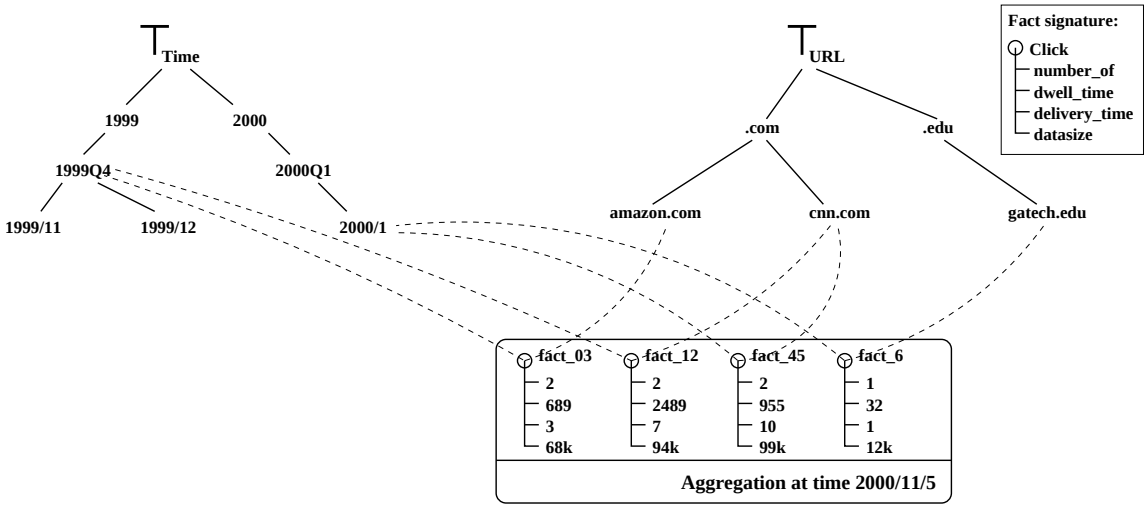

Figure 5: Query $\alpha[Time.month, URL.domain](O)$ Evaluated on the Reduced MO

Having defined the aggregate formation operator as above solves aggregation for the *availability* approach. Both the *strict* and the *LUB* approaches can be defined similarly. How aggregation is solved for the *disaggregated* approach can be studied in [13].

7 Implementation

We proceed to present a strategy for implementing the data reduction functionality described in the previous sections.

7.1 Design

The implementation strategy exploits the property that for each fact, at most one action specification, is responsible for aggregation to the fact's lowest available category type (see Section 4). Based on this, we transform the action set into a set of disjoint actions, i.e., a set where no two actions have overlapping predicates both satisfied for identical facts. Since two disjoint actions can specify aggregation to the same granularity, we group the disjoint actions on identical granularities and implement the MO using a set of subcubes, such that each subcube will represent a group of actions and have a fixed granularity. One additional subcube will have the bottom-level granularity (see Figure 6).

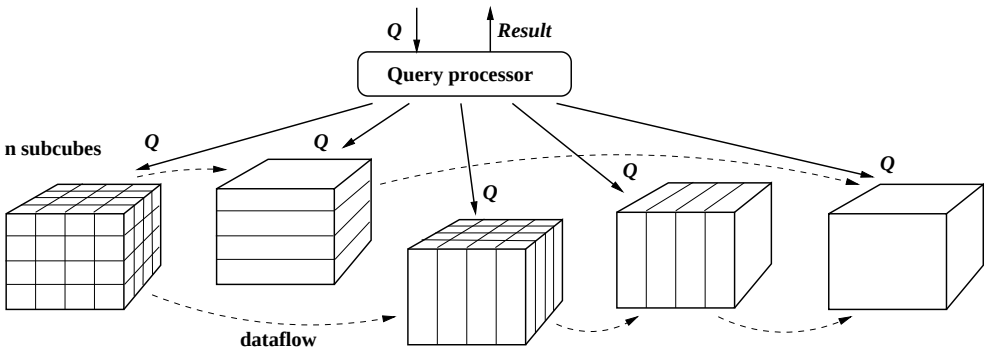

Figure 6: Architecture Using Subcubes

With this strategy, since the NOW-variable enables the aggregation of facts to higher granularities as time pases, is that synchronization of the subcubes is necessary. For the subcubes to hold facts of the correct granularities, data must be migrated between subcubes, as indicated in the figure. Another consequence is that each query must be processed against all subcubes. How the two are handled is outlined in the following subsections.

We note some important properties of the subcubes. First, all new data enter into the subcube having the bottom-level granularity. Second, subcubes only get un-synchronized due to time passing. And finally, because of the $Growing$ property of the action set, data is always moved directly from one subcube to another. This creates a natural parent-child relationship, where data is aggregated from the parent to the child cube.

EXAMPLE: Considering the data reduction specification $(\{a_1', a_2, a_6\}, \leq_\nu)$ with distinct actions below, and the MO from the ongoing example. Then each of these three distinct actions define a physical cube containing reduced aggregated data. (The predicate in a_1' is a transformation from "$\ldots \wedge NOW - 12months < Time.month \leq NOW - 6months$," i.e., from the predicate in a_1 from the ongoing example.) $a_\perp$ define a physical cube for the remaining detail facts having the bottom-level granularity.

$$a_1' = \rho(\alpha[Time.month, URL.domain]\ \sigma[URL.domain\_grp = \texttt{.com} \wedge NOW - 4quarters < Time.quarter \wedge Time.month \leq NOW - 6months](O)) \quad (41)$$

$$a_2 = \rho(\alpha[Time.quarter, URL.domain] \sigma[URL.domain\_grp = \texttt{.com} \wedge Time.quarter \leq NOW - 4quarters](O)) \quad (42)$$

$$a_6 = \rho(\alpha[Time.week, URL.domain] \sigma[URL.domain = \texttt{gatech.edu} \wedge Time.week \leq NOW - 36weeks](O)) \quad (43)$$

$$a_\perp = \rho(\alpha[Time.day, URL.url]\sigma[\neg(URL.domain\_grp = \texttt{.com} \wedge Time.month \leq NOW - 6months) \wedge \neg(URL.domain = \texttt{gatech.edu} \wedge Time.week \leq NOW - 36weeks)](O)) \quad (44)$$

All new data enters into $a_\perp$ which is the parent of both a_1' and a_6, while a_1' is the parent of a_2. □

7.2 Synchronization and Data Flow

We first explain the states of un-synchronization that can be caused by the use of the NOW-variable, then how to perform the synchronization. We assume that a fact is only out of synchronization for one level in the parent-child hierarchy. (We later explain why this is a fair assumption.)

The most frequent need for synchronization is due to either time changing or new data arriving at the warehouse. To synchronize the subcubes in these cases, first any new data must be inserted into the top-most parent subcube. Then for each parent-child combination, data might be present in the parent that satisfies the child predicate and not the parent predicate. Such data must be aggregated into the child cube. Finally, if a subcube has more parents, aggregated data can enter from different places, and more facts might exist for one cell, so that the facts within the subcube must be aggregated one final time.

An infrequent need for synchronization occurs when the data reduction specification is changed. This may necessitate the creation of new subcubes and the deletion of old ones. To synchronize in this case, first the new set of disjoint actions must be calculated, which may lead to new subcubes being created. Then the synchronization explained above can be applied, but now data is moved from all old subcubes, not only from parent cubes. Finally, all old subcubes no longer available after re-calculating the action set will be empty and should be deleted.

EXAMPLE: Assume the subcubes K_1, K_2, and K_4 in Figure 7, where K_1 and K_4 both contain facts be-

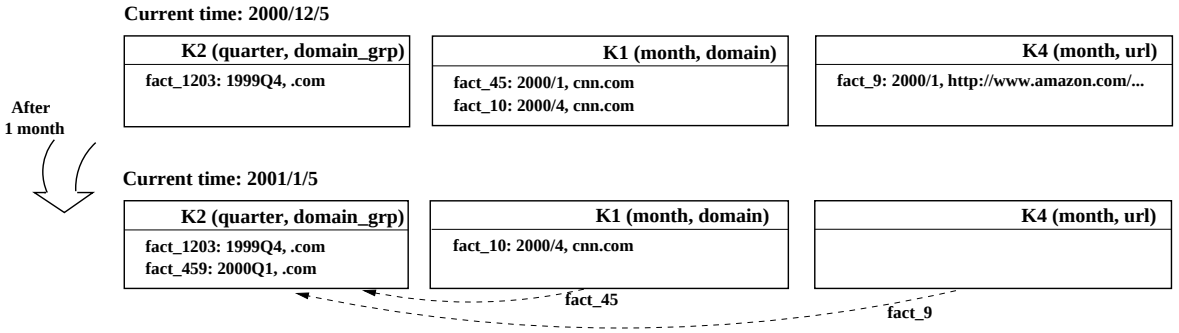

Figure 7: Synchronization from Subcubes K_1 and K_4 into K_2

tween 6 and 12 months of age, K_1 having `cnn.com`-facts and K_4 having `amazon.com`-facts. Since K_2 is the child of both K_1 and K_4, facts move from K_1 and K_4 to K_2 as time passes. For example, at time $2000/12/5$, facts $fact\_45$ and $fact\_9$ are in K_1 and K_4, respectively, and at time $2001/1/5$, they are aggregated into K_2 as $fact\_459$. □

The most frequent synchronization occurs due either to new data being bulk-loaded into the warehouse or to the value of the NOW-variable changing. In each action specification, the granularity of the NOW-variable, e.g., month, limits how frequently a cube may get un-synchronized, e.g., at most once a month. Bulk-loading is likely to happen more often. This leads to data being inserted into one subcube, so only one parent must be synchronized in this situation. Also, since insertion occurs in bulk, this synchronization is not considered a performance bottleneck, and subcubes, can fairly easily be kept synchronized.

We have assumed that the cubes would only be out of synchronization for one parent-child level at the time of synchronization. This will be true if synchronization is scheduled at the time of insertion and at least once per significant time period, the second-lowest granularity at which the NOW-variable is used in an action. For example, if the finest granularity specified is months, and the child cube has a granularity of quarters, it is sufficient to synchronize once per quarter to satisfy our assumption.

7.3 Querying Subcubes

The querying of a set of physical subcubes is briefly explained, first assuming fully synchronized cubes, and then allowing the consequences of an un-synchronized state. The idea is to evaluate the query on the subcubes, separately and in parallel, and then to combine the subresults into the final result.

In the first case we may assume m disjoint subcubes, each of which represents data with a well-specified granularity and satisfying a known predicate, and each of which are fully synchronized. A query can then be evaluated on each of the m subcubes returning up to m subresults that can be combined by an aggregation. The following example shows the structure of the subresults, and it illustrates how they are combined following the *availability* approach for aggregation.

EXAMPLE: We assume 5 synchronized subcubes $\{K_0, \ldots, K_4\}$ at time $2000/10/20$, as illustrated at the bottom of Figure 8. Compared to the ongoing example, K_1 is now split into K_1 and K_4, and 4 new facts appear, one in K_1, one in K_4, and two in K_0.

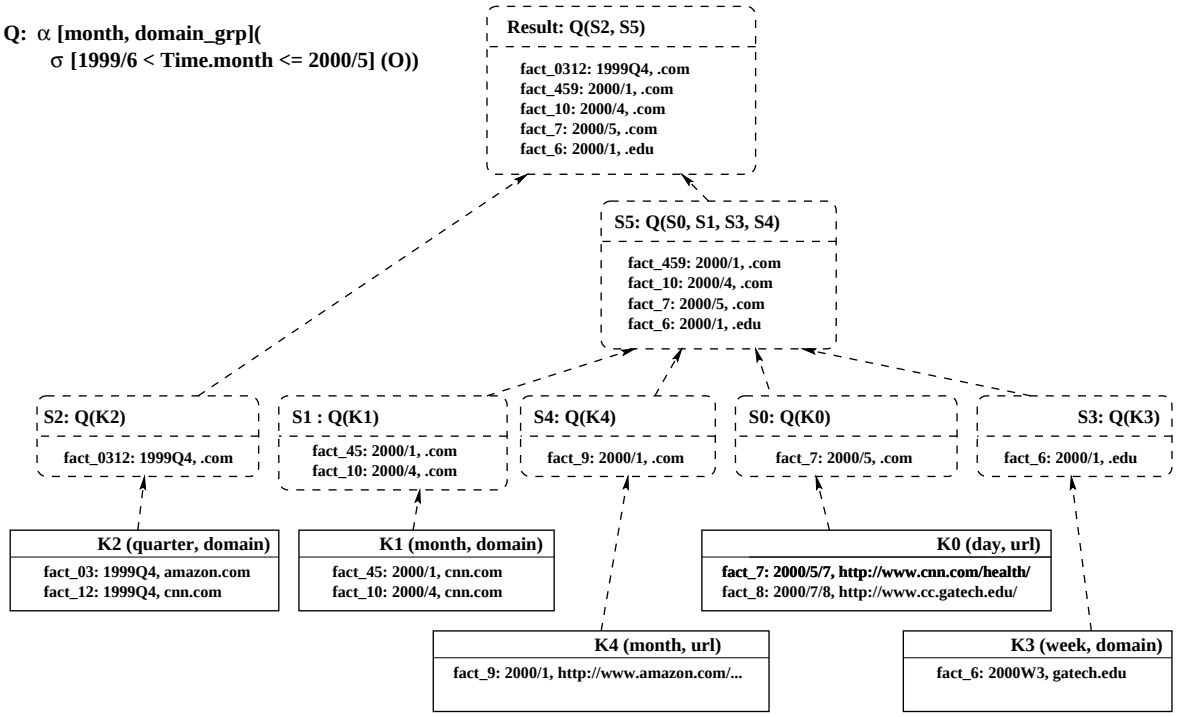

Figure 8: Evaluation plan for Query Q

The query specified in the figure is evaluated separately on each of the 5 subcubes, giving 5 subresults $S_0, S_1, \ldots, S_4$. We note that the granularity of S_2 is $(Time.quarter, URL.domain\_grp)$ since the $month$-value is no longer available in cube K_2, and that all other sub-results are aggregated to the desired granularity, but also that none of them are guaranteed to cover the entire cells of this granularity. Therefore, a final aggregation step is necessary for S_0, S_1, S_3, and S_4. Doing this, some subresults aggregate into new facts, e.g., $fact\_45$ and $fact\_9$ will aggregate to $fact\_459$ as shown in subresult S_5. Since all the usual aggregate functions used in data warehousing are distributive or can be formulated using only distributive functions, it poses no complications to do the aggregation in two steps as done here. □

Following the strategy illustrated in the example above, one query evaluation consists of m individual sub-query evaluations that can be done in parallel, and then of only a few additional aggregations and one union of disjoint answers.

When we allow the subcubes to be un-synchronized, the process becomes a little more complicated. To "see" only data of the expected granularity and satisfying the specified predicate in a cube, we must select the data satisfying this predicate from the cube itself and its parent cubes, and we must aggregate the data

to the correct granularity. Only then we can process the query on the subcube. This approach includes the synchronization of a subcube in the query process

We will illustrate the consequences on the example in Figure 8 above.

EXAMPLE: With the current time $2001/1/20$ and the un-synchronized subcubes $\{K_0, \ldots, K_4\}$ from Figure 8 at this time, we zoom in on cube K_1 which represents the distinct action a_1'.

$$a_1' : \rho(\alpha[Time.month, URL.domain]\sigma[URL.domain = \texttt{cnn.com} \wedge \\ NOW - 4quarters < Time.quarter \wedge Time.month \leq NOW - 6months](O))$$

At the current time of $2001/1/20$, $fact\_45$ no longer satisfies the predicate of a_1', but is not yet synchronized into the single child-cube of K_2. Instead both $fact\_10$ and $fact\_7$ satisfies the predicate of a_1', $fact\_10$ is present in cube K_1 but $fact\_7$ is not yet synchronized into K_1. Since K_0 is the single parent of K_1 naturally this is the only cube we need to examine for un-synchronized facts with respect to cube K_1. The situation looks like in Figure 9 where dotted lines represent subqueries and the fact sets of relevance to each subquery.

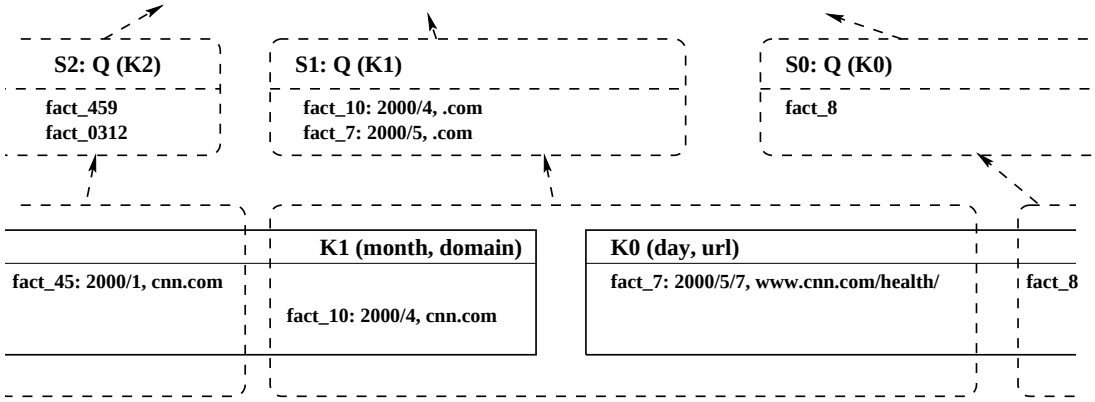

Figure 9: Evaluating the Subquery S_1 on K_1 in an Un-synchronized State

As we see, querying K_1 when this is un-synchronized in one generation forces us to query K_1 and all its immediate parents to get the subresult related to the cube K_1. Similarly when querying all other subcubes, it is necessary to include all parent subcubes. Note, that this makes a difference when querying subcubes of higher granularity than asked for, i.e., when querying K_2 both $fact\_45$ and $fact\_9$ should be part of the result having granularity $(Time.quarter, URL.domain\_grp)$. □

This principle is used in the querying strategy handling querying in the un-synchronized state. Instead of evaluating a query Q un a physical subcube, Q is evaluated on the union of the subcubes achieved by using the cube specifications on the subcube itself and all parent-subcubes, i.e., instead of evaluating Q on K_i in the synchronized state, then assuming K_i is specified by granularity G_i and predicate P_i and has the parent subcubes $\{K_j\}$, Q is evaluated on $\alpha[G_i]\sigma[P_i]\big(K_i\big) \cup \big(\bigcup_{K_j} \alpha[G_i]\sigma[P_i]\big(K_j\big)\big)$ in the un-synchronized state. Now, as in the synchronized case Q is evaluated on the union of all subresults S_i that has the granularity as specified in the query, and finally all subresults are unioned to get the final result.

With this strategy the un-synchronized state only adds the "$\alpha[G_i]\sigma[P_i]$"s when querying the single subcubes. Also, we notice that it will be sufficient to process the "$\alpha[G_i]\sigma[P_i]$"s for those subcubes having a larger granularity than asked for in the query. For all other subcubes only the selection is necessary since the facts will be aggregated afterwards.

With un-synchronized cubes, each sub-query evaluation needs to access facts from its parent subcubes. As before, this can be done in parallel and leads to disjoint subresults that are easily combined.

8 Summary and Future Research

This paper proposed a powerful and easy-to-use technique for data reduction in dimensional data warehouses. The technique is inspired by the many current data warehouses that are containing massive amounts of data and are growing rapidly, and by the often decreased interest in old, detailed data. By using the proposed technique, huge storage gains are enabled while retaining essential high-level information.

The technique uses aggregation to gradually reduce the detailed data into higher-level summarized data, taking up much less space. The data reduction is based on specifications of when data is to be aggregated to certain granularities. Following the specifications, data will progressively and automatically be reduced, e.g., over time, by repeated aggregation. Great care is taken to ensure the semantic correctness of the data reduction technique, since aggregation of a fact is inherently irreversible, and to ensure soundness of the technique. To ensure this, we present rules and properties for updating the specifications together with considerations on the operational handling of these rules. Also, techniques for the querying of reduced warehouse and detailed strategies for implementation of the data reduction technique are presented.

We believe this to be the first technique for gradual and automatic reduction of multidimensional data, e.g., over time. We also believe this paper is the first to consider the issues related to the semantic correctness of a gradual data reduction process. Additionally, the techniques for querying of multidimensional data that is gradually being aggregated are believed to be novel.

In future research, two directions are of interest. First, it is interesting to explore the efficient implementation of the technique using current data warehouse technology. Related to this, issues related to the scheduling of reduction actions and queries should be investigated. Second, the technique can be extended to explore reduction in the number of dimensions and measures, as well as to also allow for the deletion of facts.

References

[1] M. E. Adiba and B. G. Lindsay. Database Snapshots. In *Proceedings of the 6th International Conference on Very Large Databases*, pages 86–91, 1980.

[2] D. Barbará, W. DuMouchel, C. Faloutsos, P. J. Haas, J. M. Hellerstein, Y. Ioannidis, H. V. Jagadish, T. Johnson, R. Ng, V. Poosala, K. A. Ross, and K. C. Sevcik. The New Jersey Data Reduction Report. *IEEE Data Engineering Bulletin*, 20(4):3–42, 1997.

[3] S. N. Burris. *Logic for Mathematics and Computer Science* Prentice Hall, 1998.

[4] J. Clifford, C. Dyreson, T. Isakowitz, C. S. Jensen, and R. T. Snodgrass. On the Semantics of "Now" in Databases. *ACM Transactions on Database Systems*, 22(2):171–214, June 1997.

[5] C. E. Dyreson. A Bibliography on Uncertainty Management in Information Systems. In [11], pages 415–458, 1996.

[6] H. Garcia-Molina, W. Labio, and J. Yang. Expiring Data in a Warehouse. In *Proceedings of the 24th International Conference on Very Large Databases*, pages 500–511, 1998.

[7] J. Gray et al. Data Cube: A Relational Aggregation Operator Generalizing Group-By, Cross-Tab and Sub-Totals. *Data Mining and Knowledge Discovery*, 1(1):29–54, 1997.

[8] A. Gupta and I. S. Mumick (editors). *Materialized Views—Techniques, Implementations, and Applications.* The MIT Press, 1999.

[9] A. Klug. Equivalence of Relational Algebra And Relational Calculus Query Languages Having Aggregate Functions. *Journal of the ACM*, 29(3):699–717, July 1982.

[10] M. Last and O. Maimon. Automated Dimensionality Reduction of Data Warehouses. In *Proceedings of the Second International Workshop on Design and Management of Data Warehouses*, Stockholm, Sweden, 9 pages, June 2000.

[11] A. Motro and P. Smets (Eds.) *Uncertainty Management in Information Systems: From Needs to Solution.* Kluwer Academic Publishers, Boston, ISBN 0-7923-9803-3

[12] S. Owre, S. Rajan, J. M. Rushby, N. Shankar, and M. K. Srivas. PVS: Combining Specification, Proof Checking, and Model Checking. In Rajeev Alur and Thomas A. Henzinger, editors, *Computer-Aided Verification, CAV '96*, volume 1102 of *Lecture Notes in Computer Science*, pages 411–414, New Brunswick, NJ, July/August 1996. Springer-Verlag.

[13] T. B. Pedersen, C. S. Jensen, and C. E. Dyreson. A Foundation for Capturing and Querying Complex Multidimensional Data. In *Information Systems - Special Issue: Data Warehousing*, 42 pages, 2001, to appear.

[14] R. T. Snodgrass and I. Ahn. Temporal Databases. *IEEE Computer*, 19(9):35–42, September 1986.

[15] J. Skyt and C. S. Jensen. Managing Aging Data Using Persistent Views (extended abstract). In *Proceedings of the 7th International Conference on Cooperative Information Systems*, pages 132–137, September 2000.

[16] J. Skyt and C. S. Jensen. Vacuuming Temporal Databases. TIMECENTER Technical Report TR-32, `www.cs.auc.dk/TimeCenter`, Department of Computer Science, Aalborg University, 20 pages, September 1998.

[17] J. Widom (editor). Special Issue on Materialized Views and Data Warehousing. *IEEE Data Engineering Bulletin*, 18(2), June 1995.

A Formalized Example

Table 2 illustrates the database tables containing the example data for the ISP data warehouse we use throughout the paper.

Time Dimension

date_id	day	week	month	quarter	year
201	$1999/11/23$	$1999W47$	$1999/11$	$1999Q4$	1999
203	$1999/12/4$	$1999W48$	$1999/12$	$1999Q4$	1999
204	$1999/12/31$	$1999W52$	$1999/12$	$1999Q4$	1999
206	$2000/1/4$	$2000W1$	$2000/1$	$2000Q1$	2000
207	$2000/1/20$	$2000W3$	$2000/1$	$2000Q1$	2000

URL Dimension

url_id	url	domain	domain_grp
601	`http://www.cc.gatech.edu/`	`gatech.edu`	`.edu`
602	`http://www.cnn.com/`	`cnn.com`	`.com`
603	`http://www.cnn.com/health`	`cnn.com`	`.com`
604	`http://www.amazon.com/exec/obidos/tg/browse/-/465600/ref=b_tn_un/107-2047155-8802158`	`amazon.com`	`.com`

Click Fact

fact_id	date_id	url_id	number_of	dwell_time	delivery_time	datasize
$fact\_0$	201	604	1	677	2	$34k$
$fact\_1$	203	603	1	2335	5	$52k$
$fact\_2$	203	602	1	154	2	$42k$
$fact\_3$	204	604	1	12	1	$34k$
$fact\_4$	206	602	1	654	4	$47k$
$fact\_5$	206	603	1	301	6	$52k$
$fact\_6$	207	601	1	32	1	$12k$

Table 2: Example Data for an ISP Data Warehouse

Using the formalism presented in Section 3 the example MO based on these data is defined as follows below and on the following page.

$O = (\mathcal{S}, F, D, R, M)$,

$\mathcal{S} = (\mathcal{F}, \mathcal{D}, \mathcal{M})$, $\mathcal{F} = Click$, $\mathcal{D} = \{Time, URL\}$, $\mathcal{M} = \{Number\_of, Dwell\_time, Delivery\_time, Datasize\}$

$Time = (\{day, week, month, quarter, year\}, \leq_{Time}, \top_{Time}, \bot_{Time} = day)$,

$day \leq_{Time} month \leq_{Time} quarter \leq_{Time} year \leq_{Time} \top_{Time}$ and $day \leq_{Time} week \leq_{Time} \top_{Time}$

$URL = (\{url, domain, domain\_grp\}, \leq_{URL}, \top_{URL}, \bot_{URL} = url)$,

$url \leq_{URL} domain \leq_{URL} domain\_grp \leq_{URL} \top_{URL}$

$F = \{fact\_0, fact\_1, fact\_2, fact\_3, fact\_4, fact\_5, fact\_6\}$

$D = \{\text{Time}, \text{URL}\}$;

Time $= (\{\text{day}, \text{week}, \text{month}, \text{quarter}, \text{year}, \top_{\text{Time}}\}, \leq_{\text{Time}})$;

day $= \{1999/11/23, 1999/12/4, 1999/12/31, 2000/1/4, 2000/1/20\}$,

week $= \{1999W47, 1999W48, 1999W52, 2000W1, 2000W3\}$, month $= \{1999/11, 1999/12, 2000/1\}$,

quarter $= \{1999Q4, 2000Q1\}$, year $= \{1999, 2000\}$, $\top_{\text{Time}} = \{\top\}$

where

$1999/11/23 \leq_{\text{Time}} 1999/11 \wedge 1999/12/4 \leq_{\text{Time}} 1999/12 \wedge 1999/12/31 \leq_{\text{Time}} 1999/12 \wedge$
$2000/1/4 \leq_{\text{Time}} 2000/1 \wedge 2000/1/20 \leq_{\text{Time}} 2000/1 \wedge 1999/11/23 \leq_{\text{Time}} 1999W47 \wedge$
$1999/12/4 \leq_{\text{Time}} 1999W48 \wedge 1999/12/31 \leq_{\text{Time}} 1999W52 \wedge 2000/1/4 \leq_{\text{Time}} 2000W1 \wedge$
$2000/1/20 \leq_{\text{Time}} 2000W3 \wedge 1999/11 \leq_{\text{Time}} 1999Q4 \wedge 1999/12 \leq_{\text{Time}} 1999Q4 \wedge$
$2000/1 \leq_{\text{Time}} 2000Q1 \wedge 1999Q4 \leq_{\text{Time}} 1999 \wedge 2000Q1 \leq_{\text{Time}} 2000 \wedge 1999 \leq_{\text{Time}} \top \wedge 2000 \leq_{\text{Time}} \top$

URL $= (\{\text{url}, \text{domain}, \text{domain\_grp}, \top_{\text{URL}}\}, \leq_{\text{URL}})$;

url $= \{$`www.cc.gatech.edu/`, `www.cnn.com/`, `www.cnn.com/health`, `www.amazon.com/ex...`$\}$,

domain $= \{$`cnn.com`, `gatech.edu`, `amazon.com`$\}$, domain_grp $= \{$`.com`, `.edu`$\}$, $\top_{\text{URL}} = \{\top\}$

where

`www.cc.gatech.edu/` $\leq_{\text{URL}}$ `gatech.edu` $\wedge$ `www.cnn.com/health/` $\leq_{\text{URL}}$ `cnn.com` $\wedge$
`www.cnn.com/` $\leq_{\text{URL}}$ `cnn.com` $\wedge$ `www.amazon.com/ex...` $\leq_{\text{URL}}$ `amazon.com` $\wedge$
`gatech.edu` $\leq_{\text{URL}}$ `.edu` $\wedge$ `cnn.com` $\leq_{\text{URL}}$ `.com` $\wedge$ `amazon.com` $\leq_{\text{URL}}$ `.com` $\wedge$ `.edu` $\leq_{\text{URL}} \top \wedge$
`.com` $\leq_{\text{URL}} \top$

$R = \{R_{Time}, R_{URL}\}$ where

$R_{Time} = \{(fact\_0, 1999/11/23), (fact\_1, 1999/12/4), (fact\_2, 1999/12/4), (fact\_3, 1999/12/31),$
$(fact\_4, 2000/1/4), (fact\_5, 2000/1/4), (fact\_6, 2000/1/20)\}$

$R_{URL} = \{(fact\_0,$ `www.amazon.com/ex...`$), (fact\_1,$ `www.cnn.com/health/`$), (fact\_2,$ `www.cnn.com/`$),$
$(fact\_3,$ `www.amazon.com/ex...`$), (fact\_4,$ `www.cnn.com/`$), (fact\_5,$ `www.cnn.com/health/`$),$
$(fact\_6,$ `www.cc.gatech.edu/`$)\}$

$M = \{\text{Number\_of}, \text{Dwell\_time}, \text{Delivery\_time}, \text{Datasize}\}$

where

$a_{\text{Number\_of}} = \text{SUM}, a_{\text{Dwell\_time}} = \text{SUM}, a_{\text{Delivery\_time}} = \text{SUM}, a_{\text{Datasize}} = \text{SUM};$

$\text{Number\_of}(fact\_0) = 1, \text{Dwell\_time}(fact\_0) = 677, \text{Delivery\_time}(fact\_0) = 2, \text{Datasize}(fact\_0) = 34k,$
$\text{Number\_of}(fact\_1) = 1, \text{Dwell\_time}(fact\_1) = 2335, \text{Delivery\_time}(fact\_1) = 5, \text{Datasize}(fact\_1) = 52k,$
$\text{Number\_of}(fact\_2) = 1, \text{Dwell\_time}(fact\_2) = 154, \text{Delivery\_time}(fact\_2) = 2, \text{Datasize}(fact\_2) = 42k,$
$\text{Number\_of}(fact\_3) = 1, \text{Dwell\_time}(fact\_3) = 12, \text{Delivery\_time}(fact\_3) = 1, \text{Datasize}(fact\_3) = 34k,$
$\text{Number\_of}(fact\_4) = 1, \text{Dwell\_time}(fact\_4) = 654, \text{Delivery\_time}(fact\_4) = 4, \text{Datasize}(fact\_4) = 47k,$
$\text{Number\_of}(fact\_5) = 1, \text{Dwell\_time}(fact\_5) = 301, \text{Delivery\_time}(fact\_5) = 6, \text{Datasize}(fact\_5) = 52k,$
$\text{Number\_of}(fact\_6) = 1, \text{Dwell\_time}(fact\_6) = 32, \text{Delivery\_time}(fact\_6) = 1, \text{Datasize}(fact\_6) = 12k$