Structural Repairs of Multidimensional Databases*

Sina Ariyan and **Leopoldo Bertossi**†
Carleton University, School of Computer Science, Ottawa, Canada.
{mariyan,bertossi}@scs.carleton.ca

Abstract. In a multidimensional (MD) database, dimensions may be subject to semantic conditions that are not enforced by MD DBMSs or data warehouse applications. Strictness and homogeneity are possibly two of them; and are crucial for the efficiency and correctness of answering MD aggregate queries and updating materialized aggregate views. Dimensions may become inconsistent, i.e. non-strict or heterogeneous, as the result of update operations. As a methodology to restore consistency, we propose and investigate changes to the dimension schema, as an alternative to changes on the dimension instance. We introduce the notion of *minimal structural repair*, and establish that under certain conditions, a structural repair reduces the cost wrt changing the dimension instance. We also show that it allows for a correct rewriting of queries posed to the original MD model into queries in terms of the new schema. Finally, we show how query-scoped calculated members in MDX can be used to create virtual repairs that simulate structural repairs.

1 Introduction

Multidimensional databases (MDDBs) represent data in terms of *dimensions* and *fact tables*. Now, a dimension is represented as a *dimension schema*, i.e. a hierarchy (or lattice) of categories [13, 10], plus a *dimension instance* that assigns data elements to categories, organizing them in a hierarchy that parallels the category hierarchy. Facts are quantitative measures assigned to dimension instances. This MD organization allows users to aggregate data at different levels of granularity. Aggregation is the most important operation in OLAP applications. Performing fast query answering becomes crucial since queries are complex, involve aggregation and also much data [15].

Figure 1 shows the schema and instance of a *Location* dimension. The categories are: `City, State, Province, Country, All`. Here, for example, `USA, England, UK, Canada` are data elements, of the `Country` category. Categories (and elements therein) are partially ordered upwards. We say, for example, that `Country` is an *ancestor* category of `City`. If an element a is connected to an element b that belongs to a higher category, we say that a *rolls up* to b.

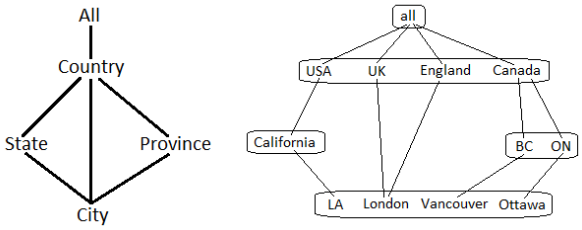


Fig. 1: Schema(left) and instance(right) of a dimension

In an ideal situation, MDDBs are expected to satisfy the *strictness* (aka. *no double counting*) and *homogeneity* (aka. *covering*) conditions. These become semantic con-

* This research was funded by an NSERC Discovery Grant and the NSERC Strategic Network on Business Intelligence (BIN), ADC02 project.

† Faculty Fellow of the IBM CAS, Toronto.

straints on dimension instances. In a strict dimension, every element of a category rolls up to at most one element in a same ancestor category.

In a homogeneous dimension, all elements in a category have the same structure. More precisely, if a category C has C' as an ancestor category, then all elements of C must roll up to some element in C'. The dimension instance in Figure 1 is not homogeneous, i.e. heterogeneous, because the element `London` does not roll up to any element in categories `State` or `Province`. It is also non-strict, because `London` rolls up to the two different elements, `England` and `UK`, in the `Country` category.

Dimension instances that do not satisfy strictness or homogeneity (or both) are said to be *inconsistent*. Hence, the dimension in Figure 1 is inconsistent. Dimensions may become inconsistent for several reasons, in particular, as the result of a poor design or a series of update operations. Inconsistent dimensions reduce soundness and efficiency of OLAP systems. In them it is not possible to assume that the *summarizability property* holds [13, 14]. In a summarizable dimension, an aggregate view defined for a parent category can be correctly derived from a set of pre-computed views for its child categories. Summarizability allows for the correct reuse of materialized aggregate views.

To restore consistency, changes have to be made to the dimension schema and/or the dimension instance. In accordance with the area of consistent query answering (CQA) in relational databases [1, 3], the resulting dimension is called a *repair* of the original one. A *minimal repair* is one that *minimally differs* from the original dimension. As expected, minimality can be defined and characterized in different ways [3]. Previous work on repairing MDDBs has focused mainly on changing the dimension instance by modifying data elements or links between them [4, 5, 6]. Repairs obtained in this way are called *data repairs*. In them, the proposed approach to removing heterogeneity is the addition of different null values in the dimension instance [14, 18].

As indicated above, inconsistency may be caused by a problematic dimension schema. Therefore, changing the dimension instance may be only a temporary solution. Future update operations will probably produce new inconsistencies. So, exploring and considering structural changes, i.e. changes in the schema, is a natural way to go. This is an idea that has been suggested in [14], to overcome heterogeneity. In this work we introduce and investigate the notion of *structural repair*, as an alternative to data repair. They modify the schema of an inconsistent dimension, to restore strictness and homogeneity.

Structural repairs restrict changes to dimension instances by allowing changes that affect only the category lattice and the distribution of the data elements in different categories. In particular, there are no "data changes" (as in data repairs), in the sense that the data elements and also the edges between them remain. In the case of a ROLAP data warehouse (with relational schemas, like *star* or *snow flake*) fewer data changes on a dimension instance directly results in fewer changes on records in the underlying database.

We establish that any minimal structural repair enables *query rewriting*: Given a query, $\mathcal{Q}$, posed to the original, inconsistent MDDB D and any of its minimal repairs, D', $\mathcal{Q}$ can be translated into a query $\mathcal{Q}'$ to be posed to D', obtaining the same results in terms of selected data elements and their respective aggregate values. In particular, if in D some summarizability properties applied or "local" homogeneity and strictness held (which, e.g. can be enforced by local semantic constraints [5]), then the query answers for the "correct" cases in D are reobtained via $\mathcal{Q}'$ from D'. Even more, the

same (correct) summarizability properties that held in D will hold in D'. Since repairs are now consistent, summarizability will be a global property in each of them. [1]

Structural repairs can be used to solve both non-strictness and heterogeneity, and have major advantages wrt to other approaches, specially when dealing with heterogeneity. On the other hand, when fixing non-strictness, structural repairs have good properties wrt the number of changes when the parent elements in a same category involved in a violation of strictness (i.e. they have a descendant in common) do not have descendants in many different categories. In such undesirable cases, changing the parent category may have to be propagated to categories belonging to its lower subgraph, causing several additional changes (cf. Section 6). Thus, a data repair or a combination of data and structural repair may result in fewer overall changes.

Structural repairs may be virtually implemented through appropriate view definitions, an idea we develop in this work. As a view definition language, we consider MDX, the language commonly used to query MDDBs [17, 19]. In this work we also show how *query-scoped calculated members* in the MDX language can be used to create virtual structural repairs. In this way, the behavior of structural repairs can be explored at query time, which is useful if a structural repair of choice is going to be materialized at some point.

This paper is organized as follows: Section 2 presents the MD data model. Section 3 formalizes the notion of (minimal) structural repair. Section 4 discusses properties of an ideal repair and shows that, by satisfying certain conditions, a minimal structural repair will have those properties. Section 5 shows how virtual structural repairs can be created in MDX. Section 6 provides a discussion of previous work, a comparison between data repairs and structural repairs, directions for future work, and some concluding remarks. Proofs of results, additional examples, and a brief introduction to MDX can be found in the Appendix.

2 The Multidimensional Model

In this work we adopt as a basis the Hurtado-Mendelzon model of MDDBs [10, 13]. However, since we will deal with changes in the schema, we will use a two-sorted structural representation of a MDDB (as in many-sorted predicate logic [8]). This will flatten out the representation and will allow us to talk about categories as elements of the structure (as opposed to sets of elements). In consequence, a *dimension* is a set-theoretic structure of the form $\mathcal{S} = \langle U, C^U, E^U, L_C^U, L_E^U, EC^U \rangle$, where C^U and E^U are unary relations, and L_C^U, L_E^U, EC^U are binary relations, all of them over the *universe* U. More precisely:

1. U is a possibly infinite and non-empty set that is the disjunct union of a set of categories and a set of data elements, say $U = U_C \cup U_E$.
2. $C^U \subseteq U_C$ is a finite, non-empty set of "active" categories. In the example above, $C^U = \{$`City, State, Province, Country, All`$\}$.
3. $E^U \subseteq U_E$ is a finite, non-empty set of "active" data elements. In the example, $E^U = \{$`LA, ..., all`$\}$.
4. L_C^U is the child/parent relationship between categories, i.e. edges between categories in the dimension schema. In the example, $L_C^U = \{$(`City`, `State`), (`State`, `Country`), ..., (`Country`, `All`)$\}$.

[1] Actually, all these claims still hold under slightly softer conditions than those imposed by the notion of minimal repair.

5. L_E^U is the child/parent relationship between data elements, i.e. edges between data elements in the dimension instance. In the example, $L_E^U = \{(\texttt{LA}, \texttt{California}), (\texttt{California, USA}), \ldots, (\texttt{Canada,all})\}$.
6. EC^U is a relation that maps data elements to their categories. In the example, $EC^U = \{(\texttt{LA,City}), (\texttt{California, State}), \ldots, (\texttt{all,All})\}$. `all` is the only "element" of `All`.[2]

From a dimension $\mathcal{S}$ as above we can define its *dimension schema structure*: $K(\mathcal{S}) := \langle U_C, C^U, L_C^U \rangle$. We can also define its *dimension instance*: $M(\mathcal{S}) := \langle U, E^U, L_E^U, EC^U \rangle$. Loosely speaking, we can say that "$M(\mathcal{S})$ is the instance for schema $K(\mathcal{S})$".

For a structure $\mathcal{S}$ to represent a valid MDM, it has to satisfy the following *basic MD conditions* (BMDCs): (below e_i and c_i represent members of the universe and R^* denotes the transitive closure of a binary relation R):

1. $C^U \cap E^U = \emptyset$. 2. $L_C^U \subseteq C^U \times C^U$. 3. $L_E^U \subseteq E^U \times E^U$. 4. $EC^U \subseteq E^U \times C^U$.
5. For all e_1, e_2, c_1, c_2: If $(e_1, e_2) \in L_E^U$ and $(e_1, c_1) \in EC^U$ and $(e_2, c_2) \in EC^U$, then $(c_1, c_2) \in L_C^U$.
6. For all c_1: If $c_1 \in C^U$, then $(c_1, c_1) \notin (L_C^U)^*$.
7. For all e_1: If $e_1 \in E^U$, then there exists c_1 with $(e_1, c_1) \in EC^U$.
8. For all c_1, c_2: If there is e with $(e, c_1) \in EC^U$ and $(e, c_2) \in EC^U$, then $c_1 = c_2$.
9. For all c_1, c_2: $c_1 = c_2$ iff, for all $e \in E^U$, it holds: $(e, c_1) \in EC^U$ iff $(e, c_2) \in EC^U$.

Notice that from these conditions it follows that 2. an 3. are proper inclusions.

Definition 1. Dimension $\mathcal{S} = \langle U, C^U, E^U, L_C^U, L_E^U, EC^U \rangle$ is: (a) *Strict* if, for all e_1, e_2, e_3, c : If $(e_1, e_2) \in (L_E^U)^*, (e_1, e_3) \in (L_E^U)^*, (e_2, c) \in EC^U, (e_3, c) \in EC^U$, then $e_2 = e_3$. (b) *Homogeneous* if, for all e_1, c_1, c_2 : If $(e_1, c_1) \in EC^U, (c_1, c_2) \in L_C^U$, then there is e_2, with $(e_2, c_2) \in EC^U$ and $(e_1, e_2) \in L_E^U$. (c) *Consistent* if it satisfies the two previous conditions; and *inconsistent* otherwise. □

3 Structural Repairs

Definition 2. A *data repair* for an inconsistent dimension $\mathcal{S} = \langle U, C^U, E^U, L_C^U, L_E^U, EC^U \rangle$ is a structure $\mathcal{S}' = \langle U, C^U, \widehat{E}^U, L_C^U, \widehat{L}_E^U, \widehat{EC}^U \rangle$ that satisfies the dimension conditions and the BMDCs,[3] and is strict and homogeneous. □

Notice that $\mathcal{S}$ and $\mathcal{S}'$ have the same universe, the same categories, and the same category links (but other relations may differ). We can see that, in order to obtain a data repair, one can only add or remove data elements, change links between elements or move elements between different categories. The following notion of structural repair (SR) is given in terms of a new MD structure, possibly with a new schema, and a mapping that establishes the correspondence between the original dimension and the new structure.

Definition 3. A *structural repair* for an inconsistent dimension $\mathcal{S} = \langle U, C^U, E^U, L_C^U, L_E^U, EC^U \rangle$ is a pair $\langle \mathcal{S}', g \rangle$, with $\mathcal{S}'$ a structure $\langle U, \widehat{C}^U, E^U, \widehat{L}_C^U, L_E^U, \widehat{EC}^U \rangle$, with the following properties: (a) $\mathcal{S}'$ is strict and homogeneous. (b) Elements cannot move

[2] We assume that in MDM all dimensions have this top category `All` with the single element `all`, to which all the other elements roll up.

[3] In the following this will be implicity assumed.

between categories that exist both in $\mathcal{S}$ and $\mathcal{S}'$: For all e, c_1, c_2, if $(e, c_1) \in EC^U$, $(e, c_2) \in \widehat{EC}^U$, $c_1 \neq c_2$, then $\{c_1, c_2\} \not\subseteq (C^U \cap \widehat{C}^U)$. (c) Any new category in $\mathcal{S}'$ must have at least one data element. (d) $g : C^U \to 2^{\widehat{C}^U}$, the *schema mapping*, is a total function that maps each category of $\mathcal{S}$ to a set of categories in $\widehat{C}^U$ (which is finite in $\mathcal{S}'$). (e) If $c' \in g(c)$, then c and c' share at least one data element. □

The role of g is to establish a relationship between the schemas of $\mathcal{S}$ and $\mathcal{S}'$. Notice that, for each category c, the set of its "elements", i.e. $\{e \mid (e, c) \in EC^U\}$, may be $\subseteq$-incomparable with $\{e \mid (e, c') \in \widehat{EC}^U$ and $c' \in g(c)\}$).

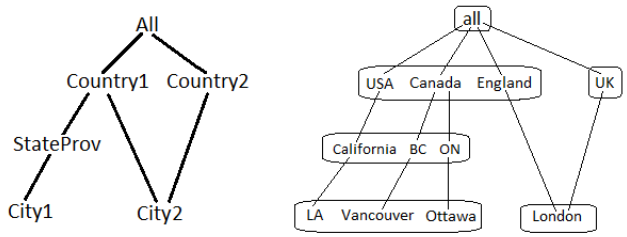


Fig. 2: An SR for the dimension of Figure 1

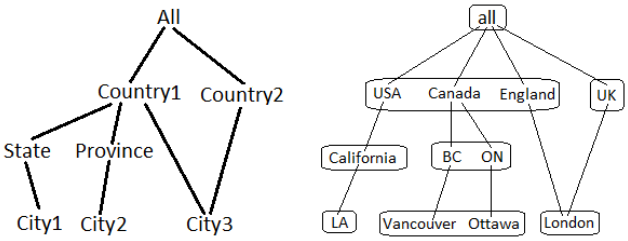


Fig. 3: Another SR for the dimension of Figure 1

A structural repair has the same domain, including categories and elements, as the initial dimension. However, the finite sets of active categories, i.e. $C^U, \widehat{C}^U$, resp., may be different. On the other side, the finitely many active elements are the same. Condition (b) in Definition 3 ensures that in a structural repair, data elements move from one category to another only as a result of changes made to the dimension schema (splitting or merging categories of $\mathcal{S}$).[4]

Figures 2 and 3 show two of the possible structural repairs for the inconsistent dimension of Figure 1. They show that strictness forces us to put elements `England` and `UK` in different categories. On the other hand, homogeneity forces us to either merge categories `State` and `Province` or isolate element `LA` in a new category.

Proposition 1. Every inconsistent dimension has a structural repair. □

This proposition can be proved by using a simple structural repair that essentially creates a new one-element category for each element in the dimension instance, and also links between the newly created categories whenever their single elements are connected in the original instance. Figure 4 illustrates this kind of structural repair as a third repair for the instance in Figure 1.

[4] *Isomorphic* structural repairs, i.e. that differ only in the names of active categories, will treated as being the same repair.

Example 1. The following mapping is a schema mapping between the dimension of Figure 1 and the structural repair of Figure 2:
$g:$ $\texttt{City} \mapsto \{\texttt{City1}, \texttt{City2}\}, \texttt{State} \mapsto \{\texttt{StateProv}\}, \texttt{Province} \mapsto \{\texttt{StateProv}\},$
$\texttt{Country} \mapsto \{\texttt{Country1}, \texttt{Country2}\}, \texttt{All} \mapsto \{\texttt{All}\}.$ □

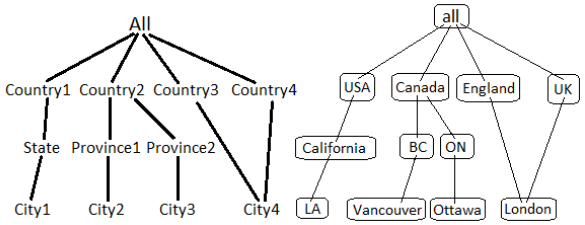


Fig. 4: Another SR for instance in Figure 1

Now we define minimal structural repairs (MSRs), as preferred repairs among the structural repairs. This requires comparing structural repairs. The *data movement set*, defined next, has useful properties for enabling this comparison (cf. Section 4).

Definition 4. Let $\langle \mathcal{S}', g \rangle$ be an SR for dimension $\mathcal{S}$, and c a category of $\mathcal{S}$. (a) The *data movement set* of category c is defined by:

$$DMSet_g(c) = \{e \mid (e, c) \in EC^U\} \ \triangle \ \bigcup_{c' \in g(c)} \{e \mid (e, c') \in \widehat{EC}^U\},$$

where $\triangle$ denotes the symmetric difference of two sets. (b) The *overall data movement set* between $\mathcal{S}$ and $\mathcal{S}'$ is $DMSet_g(\mathcal{S}, \mathcal{S}') := \bigcup_{c \in C^U} DMSet_g(c)$. □

Example 2. (example 1 cont.) For the schema mapping in the example: $DMSet_g(\texttt{City}) = \emptyset$, $DMSet_g(\texttt{State}) = \{\texttt{BC}, \texttt{ON}\}$, $DMSet_g(\texttt{Province}) = \{\texttt{California}\}$, $DMSet_g(\texttt{Country}) = \emptyset$, $DMSet_g(\texttt{All}) = \emptyset$. □

Intuitively, an MSR is a new dimension that is obtained by applying a minimal set of changes to the schema of an inconsistent dimension. Inspired by the notion of *prioritized minimization* [16], we propose to minimize both data movement and the changes in the set of categories, but assigning higher priority to minimizing the former.

Definition 5. For a dimension $\mathcal{S}$ and two SRs $\langle \mathcal{S}'_1, g_1 \rangle$ and $\langle \mathcal{S}'_2, g_2 \rangle$:

$\langle \mathcal{S}'_1, g_1 \rangle \leq_{\mathcal{S}} \langle \mathcal{S}'_2, g_2 \rangle$ iff $DMSet_{g_1}(\mathcal{S}, \mathcal{S}'_1) \subseteq DMSet_{g_2}(\mathcal{S}, \mathcal{S}'_2)$ and

$$DMSet_{g_1}(\mathcal{S}, \mathcal{S}'_1) = DMSet_{g_2}(\mathcal{S}, \mathcal{S}'_2) \Rightarrow (C^{\mathcal{S}} \triangle C^{\mathcal{S}'_1}) \subseteq (C^{\mathcal{S}} \triangle C^{\mathcal{S}'_2}).$$

Here, $C^{\mathcal{S}}$, $C^{\mathcal{S}'_1}$ and $C^{\mathcal{S}'_2}$ denote the finite sets (C^U) of active categories for structures $\mathcal{S}$, $\mathcal{S}'_1$ and $\mathcal{S}'_2$, respectively. □

Definition 6. $\langle \mathcal{S}'_1, g_1 \rangle$ is a *minimal structural repair* (MSR) of dimension $\mathcal{S}$ iff it is an SR of $\mathcal{S}$ and there is no other SR $\langle \mathcal{S}'_2, g_2 \rangle$ for $\mathcal{S}$, such that $\langle \mathcal{S}'_2, g_2 \rangle <_{\mathcal{S}} \langle \mathcal{S}'_1, g_1 \rangle$. (Here, as expected, $u <_{\mathcal{S}} v$ means $u \leq_{\mathcal{S}} v$, but not $v \leq_{\mathcal{S}} u$.) □

Example 3. It can be shown that the structural repair of Figure 3 is an MSR for the inconsistent dimension of Figure 1, with the following schema mapping:
$g_1:$ $\texttt{City} \mapsto \{\texttt{City1}, \texttt{City2}, \texttt{City3}\}, \texttt{State} \mapsto \{\texttt{State}\}, \texttt{Province} \mapsto \{\texttt{Province}\},$
$\texttt{Country} \mapsto \{\texttt{Country1}, \texttt{Country2}\}, \texttt{All} \mapsto \{\texttt{All}\}.$
On the other hand, the structural repair of Figure 2 is not an MSR for any possible schema mapping. This is because for categories `State` and `Province` in the

original dimension, there is no category (or set of categories) in this repair that contains exactly the data elements that belong to those two categories. Therefore, for any mapping g_2, $DMSet_{g_2}(\texttt{State}) \neq \emptyset$ and $DMSet_{g_2}(\texttt{Province}) \neq \emptyset$. As a result, $DMSet_{g_2}(\texttt{State}) \subsetneqq DMSet_{g_1}(\texttt{State})$. According to the definition of MSR, g_2 cannot be minimal. □

Proposition 2. An inconsistent dimension always has a minimal structural repair. □

The following example shows that an inconsistent dimension may have multiple MSRs.

Example 4. The structural repair of Figure 5, that is obtained by swapping the data elements `England` and `UK` in the structural repair of Figure 3, shows another minimal structural repair for the inconsistent dimension of Figure 1. These two minimal repairs are not isomorphic. □

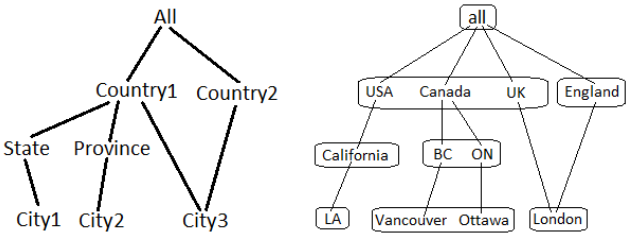


Fig. 5: Another minimal SR for the dimension of Figure 1

If the notion of MSR is applied to a consistent dimension $\mathcal{S}$, then $\langle \mathcal{S}, id \rangle$ is one of its MSRs, with $id : c \mapsto \{c\}$, but not necessarily the only one. This is because we can have categories with no members in a consistent dimension, and changing the links that are attached to them results in different minimal repairs. To ensure that a consistent dimension is its only MSR, we have to add an extra condition in the definition of minimal repair, to restrict changes on the set of category links, i.e. on L_C^U.

Our next result states two properties of MSRs that will be useful in Section 4.

Theorem 1. (a) If $\langle \mathcal{S}', g \rangle$ is an MSR of dimension $\mathcal{S}$, then for every category c in $\mathcal{S}$, $DMSet_g(c) = \emptyset$. (b) In an MSR, if $g(c) = \{c_1, \ldots, c_n\}$, then, for all e with $(e, c) \in EC^U$, there is a c_i, such that $(e, c_i) \in \widehat{EC}^U$. □

Notice that the inverse of the implication in (a) does not necessarily hold. The example in Figure 4 shows an SR with an empty data movement set that is not minimal.

4 Properties of Structural Repairs

An OLAP system is usually modeled by a three level architecture consisting of a data warehouse server, a query client and a data cube engine [7]. The performance of an OLAP system is based on how it performs at each level. Any repair is expected to remove inconsistencies of a dimension, while maintaining good properties at each of the three levels. From this point of view, an ideal repair should satisfy the following requirements:

1. Changes to the underlying database forced by the transition from the original dimension to the repair should be as few as possible.

2. Queries posed to the original instance should be rewritable as queries to the repair. And the new queries should return similar outputs in terms of selected data elements and their respective aggregate values (cf. Section 4.1).
3. If the original structure allows summarizations over existing cube views, to compute new cube views, similar summarizations should also be possible in the repaired dimension (cf. Section 4.2).

By definition, structural repairs restrict changes to data so that elements and links in the dimension instance do not change at all. Therefore, the first requirement is strictly satisfied by the use of structural repairs. In Sections 4.1 and 4.2 we will focus on how structural repairs perform wrt the two other requirements.

4.1 Rewritability of cube views

The most common aggregate queries in DWHs are those that perform grouping by the values of a set of attributes (i.e. categories), and return a single aggregate value per group. These aggregate queries are also known as *cube views* [12]. The aggregation is achieved by upward navigation through a category path, which is captured by the notion of *roll-up relation*. More precisely, the roll-up relation between categories c_i and c_j, denoted $\mathcal{R}_{c_i}^{c_j}$, is the relation $\{(e_i, e_j) \mid (e_i, c_i) \in EC^U$ and $(e_j, c_j) \in EC^U$ and $(e_i, e_j) \in (L_E^U)^\star\}$.

Now, a cube view (query) at *granularity* $G = \langle c_1, ..., c_n \rangle$ (a list of active categories) is denoted by CV_G, and is expressed as:

```
SELECT c_j, ... c_n, f(a)
FROM T, R_{b_j}^{c_j}, ... R_{b_n}^{c_n}
WHERE conditions
GROUP BY c_j, ... c_n
```

Here, $\mathtt{c}_j, \ldots \mathtt{c}_n$ are attributes (categories) of the roll-up relations $\mathtt{R}_{b_j}^{c_j}, \ldots \mathtt{R}_{b_n}^{c_n}$ (where each b_i is a bottom-level category under c_i), T is the fact table that associates numerical values (measures) to elements in b_i, and f is one of `min, max, count, sum, avg`, applied to fact measure a. The result of this cube view, denoted $result(CV_G)$, is the set of tuples $\{t_1, ..., t_r\}$ with $t_k = \langle e_{jk}, ..., e_{nk}, a_k \rangle$. The non-aggregate portion $\langle e_{jk}, \ldots, e_{nk} \rangle$ of tuple t_k contains the values for the attributes in the SELECT clause (which are the same as in the GROUP BY). Here, e_{ik} is a data element that belongs to category c_i, and a_k is the value of aggregate function f for this tuple.

Definition 7. A cube view CV_G on categories of dimension $\mathcal{S}$ is *rewritable* on an SR $\mathcal{S}'$, if there exists a cube view $\widehat{CV_{G'}}$ on categories of $\mathcal{S}'$, such that $result(CV_G) = result(\widehat{CV_{G'}})$. $\widehat{CV_{G'}}$ is called a *rewriting* of CV_G in $\mathcal{S}'$. □

Theorem 2. A cube view CV_G at granularity $G = \langle c_1, ..., c_n \rangle$ on dimension $\mathcal{S}$ is rewritable over a structural repair $\langle \mathcal{S}', g \rangle$ if the data movement set for categories $c_1, \ldots, c_n$ is empty. In particular, cube views are always rewritable over MSRs. □

Notice that in the theorem the inverse implication may not hold: Even for repairs with non-empty data movement set, some queries may be rewritable.

Example 5. Consider the dimension of Figure 1 and a cube view (a) below for this dimension. Assume that Facts is the fact table, and Sales is a fact measure. This cube view is not rewritable on the structural repair of Figure 2, because $DMSet_g$(Province) $\neq \emptyset$. On the other hand, for the structural repair of Figure 3 (that has empty data movement set for all categories), the cube view in (b) below is a rewriting. It is obtained by

replacing $\mathcal{R}^{\texttt{Province}}_{\texttt{City}}$ by $\mathcal{R}^{\texttt{Province}}_{\texttt{City2}}$.

```
(a) SELECT Province, SUM(Sales)      (b) SELECT Province, SUM(Sales)
    FROM Facts, R^Province_City          FROM Facts, R^Province_City2
    GROUP BY Province                    GROUP BY Province
```
□

4.2 Summarizability and structural repairs

A common technique for speeding up OLAP query processing is to pre-compute some cube views and use them for the derivation (or answering) of other cube views. This approach to query answering is correct under the so-called summarizability property (cf. Example 6). It is globally guaranteed by the consistency conditions on the dimension schema. However, in the case of inconsistency, there could still be localized summarizability. Either way, ideally these summarizability properties should be preserved under structural repairs.

Example 6. Consider the dimension of Figure 1 and a pre-computed cube view $CV_{\texttt{Country}}$ in (a) below for category Country. Also, consider cube view (b) below that uses $CV_{\texttt{Country}}$ to derive aggregate values for category All:

```
(a) SELECT Country, SUM(Sales) AS S   (b) SELECT All, SUM(S)
    FROM Facts, R^Country_City            FROM CV_Country, R^All_Country
    GROUP BY Country                      GROUP BY All
```

This derivation will produce correct results, only if category All is summarizable from category Country. In the dimension of Figure 1, this summarizability property does not hold. This is because in the instance of this dimension, element London will be considered twice as a descendant of all (via England and UK). □

Cube view CV_{G_1} at granularity $G_1 = \langle c_1, ..., c_j \rangle$ *depends on* cube view CV_{G_2} at granularity $G_2 = \langle c_k, ..., c_n \rangle$ iff CV_{G_1} can be answered using the result of CV_{G_2} [9]. For this to hold, each c_i in G_1 must be summarizable from G_2 (or a subset of G_2). A formal definition of category summarizability is given in [13].

Definition 8. An SR $\langle \mathcal{S}', g \rangle$ for dimension $\mathcal{S}$ *preserves summarizability* of category c over categories $\{c_1, ..., c_n\}$ (all in $\mathcal{S}$) iff every $c' \in g(c)$ is summarizable from $\bigcup_{k=1}^{n} g(c_k)$ in $\mathcal{S}'$. □

Theorem 3. An SR $\langle \mathcal{S}', g \rangle$ for dimension $\mathcal{S}$ preserves summarizability of category c over categories $\{c_1, ..., c_n\}$ if $DMSet_g(c) = DMSet_g(c_1) = \cdots = DMSet_g(c_n) = \emptyset$. In particular, minimal structural repairs preserve all the summarizability properties. □

Example 7. For the inconsistent dimension of Figure 1, consider the structural repair of Figure 3, with the schema mapping

g : City $\mapsto$ {City1, City2, City3}, State $\mapsto$ {State}, Province $\mapsto$ {Province}, Country $\mapsto$ {Country1, Country2}, All $\mapsto$ {All}.

Since g has an empty data movement set for every category, we expect it to preserve summarizability properties that hold in the original dimension. For example, Province is summarizable from City in the dimension in Figure 1. In the MSR in Figure 3, Province is summarizable from {City1, City2, City3}. □

Notice in the previous example that, in addition to the preservation of existing summarizability properties, the structural repair is also adding extra summarizibility properties. For instance, category `All` was not summarizable from `City` through category `Country` in the original dimension, but it is now summarizable from {`City1`, `City2`, `City3`} through category `Country1`. This is achieved through the newly imposed strictness and homogeneity conditions.

5 Virtual Repairs in MDX

An important problem that needs to be addressed is how to implement structural repairs in a way that is transparent to users of MDDBs. As discussed in previous sections, one advantage of using structural repairs instead of data repairs is that the former remove inconsistencies without having to change the underlying database.

One of the interesting features of the MDX query language is the ability to define *query-scoped calculated members*. A query-scoped calculated member is a virtual object defined in terms of other existing objects by means of a query. For example, a new category (or data element) in terms of other existing categories (data elements, resp.), etc. They are generated using the `WITH` construct in MDX:

`WITH` {`MEMBER`|`SET`} *MemberIdentifier* `AS` '*member-formula*' `SELECT ...`

This template creates a new calculated member (or set of members) that is defined by the MDX expression '*member-formula*'. {`MEMBER`|`SET`} indicates if the newly created object is an element or a category, resp.

Using this feature, structural repairs can be virtually created, without any need for materializing new data or accessing the database in order to generate database views. This is specially useful if the structural repair of choice is going to be materialized at some point, after some exploratory analysis.

Example 8. The following MDX query creates a query-scoped calculated category `StateProv`, as the union of categories `State` and `Province`. It then selects the summarized value of measure `Sales` for the members in this new category. Here, `SalesCube` is the base cube (or fact table):

```
WITH SET [StateProv] AS
    '[State].MEMBERS + [Province].MEMBERS'
SELECT [Measures].[Sales] ON AXIS(0),
     [StateProv] ON AXIS(1)
FROM [SalesCube]
```
□

Structural repairs (or portions of them) can be generated using this form of view definitions, as virtual repairs. In them, new categories are defined as calculated members. Virtual repairs can be used to test how structural changes will affect existing queries before making the transition to the new MD model.

Example 9. (example 8 cont.) Consider the structural repair of Figure 2. The given MDX query defines `StateProv`, the new category in the structural repair of Figure 2. It contains the elements in the union of categories `State` and `Province`. □

6 Discussion and Conclusions

Research on repairing inconsistent dimensions has focused mainly on modifying data elements. Here we introduced and investigated the notion of structural repair, as an

alternative to data repair. Structural repairs restrict changes to the dimension instance by only allowing changes that affect the dimension schema and the organization of elements in different categories. We defined the notion of minimal structural repair and proved that, by having a minimal data movement set, it enables the rewriting of queries posed to the original multidimensional model as queries to be posed to a repair. The rewritten query the same results in terms of selected data elements and their respective aggregate values.

The notion of (data) repair was introduced in [4], including a notion of *consistent answer* in the spirit of consistent query answering (CQA) in relational databases [3]. Data repairs are generated by adding and removing links between data elements, for restoring strictness on already homogeneous dimensions. Inspired by [2], consistent answers to aggregate queries on non-strict dimensions are characterized in [4] in terms of a smallest range that contain the usual (numerical) answers obtained from the minimal data repairs. In [6], preliminary work on the effect of non-strictness on aggregate queries from homogeneous dimensions was carried out. The focus is on finding ways to retrieve consistent answers even when the MDDB is inconsistent. To restore consistency, links and elements in the dimension instance are deleted in a minimal way.

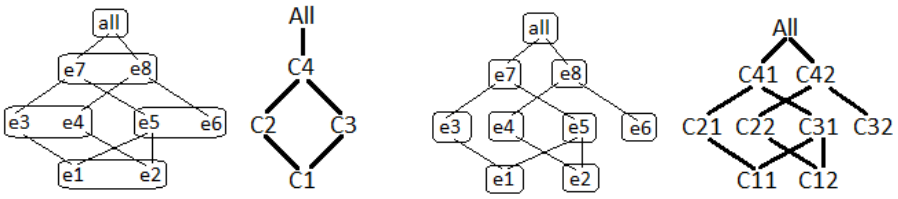
In [5], repairs of heterogeneous or non-strict dimensions (or both) are obtained by insertions and deletions of links between elements. Adding a link in the instance to remove heterogeneity may cause non-strictness in parts of the instance that did not have any problems before. Therefore, this has to be done in an iterative manner until everything is fixed (usually after many changes). Logic programs with stable model semantics were proposed to represent and compute minimal dimension repairs.

In [18], methodologies and algorithms are given for transforming inconsistent dimensions into strict and homogeneous ones. The dimension instance is changed (as opposed to the schema). Problems caused by inconsistent dimensions are fixed by introducing `NULL` members as new data elements. In [14], the authors study the implications of relaxing the homogeneity condition on MDDBs. They also argue that the addition of `NULL` values (elements) and changes to the schema provide a solution to structural heterogeneity.

An important drawback of data repairs is that they manipulate the dimension instance, i.e. the data. In case the MDDB is implemented on a relational DB, this results in changes in records, which users may find undesirable. In contrast, structural repairs can be implemented without the need to change the underlying database, through view definitions.

Another problem with data repairs is that resolving heterogeneity may require introducing redundant `NULL` data values in the dimension instance. Those values have to be proliferated along the path of ancestor categories. The alternative approach for removing heterogeneity that is based on insertions of new links has the problem of possibly introducing new sources of non-strictness, which have to be resolved. This may introduce multiple changes in the dimension instance.

Fixing non-strictness via structural repairs has good properties in terms of the number of changes when the parent elements in a same category involved in a violation of strictness (i.e. they have a descendant in common) do not have descendants in many different categories. In the opposite case, changing the parent category may have to be propagated to categories belonging to its lower subgraph, causing several additional changes, as shown in the next example.

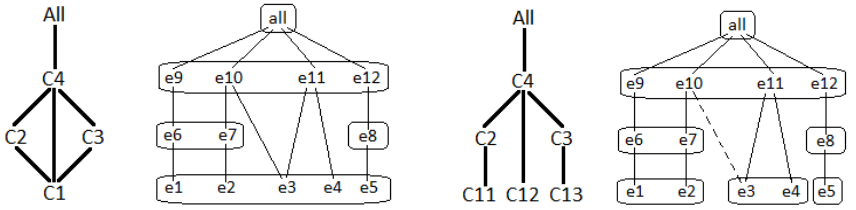


(a) A non-strict dimension (b) SR causing multiple changes

Fig. 6: Structural repairs and non-strictness

Example 10. Consider the non-strict dimension of Figure 6a, where element `e2` rolls up to two different elements, `e7` and `e8`, in category `C4`. Here, `e7` and `e8` have descendants from three different categories in their subgraph. As shown in Figure 6b, a structural repair forces many changes in the dimension schema. Using a data repair or a combination of a data and a structural repair may result in fewer overall changes. □

Inconsistencies can be resolved by combining (the ideas behind) structural and data repairs. In some cases, that still have to be properly characterized, taking advantage of both approaches for resolving different local inconsistencies may produce better results in terms of number of changes. This is illustrated in the next example.



(a) An inconsistent dimension (b) A repair that changes the schema and instance

Fig. 7: Combining structural repairs and data repairs

Example 11. Figure 7a shows an inconsistent dimension. Restriction to pure data repairs will require applying multiple data and link changes. One way is to add at least five new and different `NULL` elements and ten new links, to resolve heterogeneity; and also remove the link between `e3` and `e10`, to resolve non-strictness. On the other hand, using a pure structural repair requires introducing at least four new categories. As can be seen in Figure 7b, a combination of structural and data changes provides a solution that involves fewer changes (adding two categories in the dimension schema and removing a single link in the dimension instance). □

There are still many issues to be investigate around structural repairs, and also around their non-trivial combination with data repairs. They are the subject of ongoing research. In this work we have not developed a notion of consistent answers to queries. This is left for future research.

Our notion of a minimal structural repair can be extended with priorities imposed on certain categories (or different levels in the lattice) in the dimension schema. Higher priorities can be assigned to categories that are queried more often or contain data elements that are more sensitive to change. Multidimensional integrity constraints on the schema [11] could also be considered in the definition.

In this work we have also showed how query-scoped calculated members in MDX can be used to create virtual structural repairs. This can be seen as a way of implementing view-based (structural) repairs. We have experimented with this approach on top of the IBM Cognos 8 Business Intelligence system, which is also being used for testing the performance of structural repairs. Reporting the results is part of the future work.

References

[1] M. Arenas, L. Bertossi and J. Chomicki. Consistent Query Answers in Inconsistent Databases. In *Proc. ACM PODS'99, ACM Press*, 1999, pp. 68-79.

[2] M. Arenas, L. Bertossi, J. Chomicki, X. He, V. Raghavan and J. Spinrad. Scalar Aggregation in Inconsistent Databases. *Theoretical Computer Science*, 2003, 296(3):405-434.

[3] L. Bertossi. Consistent Query Answering in Databases. *ACM SIGMOD Record*, June 2006, 35(2):68-76.

[4] L. Bertossi, L. Bravo and M. Caniupan. Consistent Query Answering in Data Warehouses. In *Proc. 3rd Alberto Mendelzon International Workshop on Foundations of Data Management (AMW'09)*, CEUR Proc. Vol. 450, 2009.

[5] L. Bravo, M. Caniupan and C.A. Hurtado. Logic Programs for Repairing Inconsistent Dimensions in Data Warehouses. In *Proc. Alberto Mendelzon International Workshop on Foundations of Data Management (AMW'10)*, CEUR Proc. Vol. 619, 2010.

[6] M. Caniupan. Handling Inconsistencies in Data Warehouses. In *Current Trends in Database Technology*, Springer LNCS 3268, 2004, pp. 166-176.

[7] D.W. Cheung, B. Zhou, B. Kao, H. Lu, T.W. L and H.F. Ting. Requirement-Based Data Cube Schema Design. In *Proc. CIKM'99*, 1999, pp. 162-169.

[8] H. Enderton. *A Mathematical Introduction to Logic*. 2nd Edition, Academic Press, 2001.

[9] V. Harinarayan, A. Rajaraman and J. Ullman. Implementing Data Cubes Efficiently. In *Proc. SIGMOD'96*, ACM Press, 1996.

[10] C. Hurtado. Structurally Heterogeneous OLAP Dimensions. PhD Thesis, Computer Science Department, University of Toronto, 2002.

[11] C. Hurtado and A. Mendelzon. OLAP Dimension Constraints. In *Proc. PODS'02*, ACM Press, 2002, pp. 169-179.

[12] C. Hurtado and C. Gutierrez. Computing Cube View Dependences in OLAP Datacubes. In *Proc. 15th International Conference on Scientific and Statistical Database Management*, 2003, pp. 33-42.

[13] C. Hurtado, C. Gutierrez and A. Mendelzon. Capturing Summarizability With Integrity Constraints in OLAP. *ACM Transactions on Database Systems*, 2005, 30(3):854-886.

[14] C. Hurtado and C. Gutierrez. Handling Structural Heterogeneity in OLAP. In *Data Warehouses and OLAP: Concepts, Architectures and Solutions*, R. Wrembler and C. Koncilia (eds.), Idea Group, Inc. 2006.

[15] W. Inmon. *Building the Data Warehouse*. 4th Edition, Wiley, 2005.

[16] V. Lifschitz. Circumscription. In *Handbook of Logic in AI and Logic Programming*, Vol. 3, Oxford University Press, 1994, pp. 298-352.

[17] Microsoft Corporation. Multidimensional Expressions (MDX) Reference. Available at http://msdn.microsoft.com/enus/library/ms145506.aspx, 2008.

[18] T.B. Pedersen, C.S. Jensen, C.E. Dyreson. Practical Pre-aggregation in Online Analytical Processing. In *Proc. VLDB'99*, 1999, pp. 663-674.

[19] G. Spofford, S. Harinath, C. Webb, D.H. Huang, F. Civardi. *MDX Solutions: With Microsoft SQL Server Analysis Services 2005 and Hyperion Essbase*. Wiley, 2006.

A Appendix

A.1 Proofs

Proof of Proposition 1: We can always build a structural repair with a dimension schema that is the exact mirror of its dimension instance (i.e. in the schema of this structural repair there exists one category for every data element and one link for each data link in the dimension instance that connects the corresponding categories). Figure 4 shows such a structural repair for the inconsistent dimension of Figure 1. It can easily be shown that this new dimension is strict and homogeneous. First of all, every category in the new dimension contains exactly one element, and therefore this dimension is obviously strict. On the other hand, we know that links in the dimension schema have been added according to the links that exist between corresponding elements in the dimension instance. Therefore, whenever a category c_1 rolls up to category c_2, the single data element that belongs to c_1 also rolls up to the single element that belongs to c_2 which proves homogeneity. □

Proof of Proposition 2: This can be readily obtained from Proposition 1. In the worst case, a structural repair with a dimension schema that is the exact mirror of its dimension instance will be the minimal structural repair. □

Proof of Theorem 1: (a) As in the proof of Proposition 1, consider a structural repair with a dimension schema that is the exact mirror of its dimension instance (lets call it S_{mirror}). Also consider schema mapping g_m that assigns to each category c of $\mathcal{S}$ the set of categories $\{c' |\ c'$ is a category of S_{mirror} and c contains as one of its members, the single data element belonging to $c'\}$. Clearly, any category c of the originally inconsistent dimension will contain the exact same set of elements as $g(c)$ and therefore, $DMSet_{g_m}(c) = \emptyset$.

Now, we can prove the Theorem by contradiction. Lets assume there exists a category c in $\mathcal{S}$ where $DMSet_g(c) \neq \emptyset$. Therefore, $DMSet_g(S') \neq \emptyset$ and $DMSet_g(S')$ will be a proper subset of $DMSet_g(S_{mirror})$ which is a contradiction based on the definition of minimal structural repairs (Definition 6).

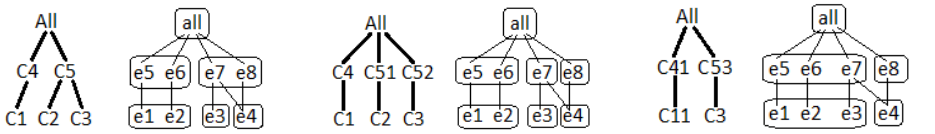
(b) This can easily be obtained from part (a) and Definition 4. If there existed an element e with no c_i, such that $(e, c_i) \in \widehat{EC}^U$, then e would belong to $DMSet_g(c)$. □

Proof of Theorem 2: It suffices to show that every category in G can be regenerated by a simple operation on categories of the structural repair. if this can be done, we can rewrite CV_G by simply replacing every occurrence of any c_i with the statement that regenerates it. Consider c_i' as a new category, with scope of a query, that contains $\{e|(e, c') \in \widehat{EC}^U\}$ as its member elements and serves as a new category that replaces c_i. Since $DMSet_g(c_i) = 0$, it readily follows from Definition 4 that c_i' will have the exact same members as c_i. Therefore, cube view CV_G can be rewritten in any query language that allows the union operation on data elements. □

Proof of Theorem 3: Based on [13], the necessary and sufficient condition for category c to be summarizable from categories $\{c_1, ..., c_n\}$ is that all elements of bottom categories that roll up to some element in c, also roll up to exactly one element that belongs to $\{c_1, ..., c_n\}$ (disjointness and completeness). We want to prove that every $c' \in g(c)$ is

summarizable from $\bigcup_{k=1}^{n} g(c_k)$ in $\mathcal{S}'$. Let's assume e_{base} is an element from a bottom category that rolls up to some element in $c' \in g(c)$ (let's call it e_{anc}). Clearly, e_{anc} also belongs to c since $DMSet_g(c) = \emptyset$. We already know that c is summarizable from $\{c_1, ..., c_n\}$, therefore e_{base} rolls up to exactly one element in $\{c_1, ..., c_n\}$ (let's call it e_{mid}). Again, since $DMSet_g(c_1) = ... = DMSet_g(c_n) = \emptyset$, e_{mid} has to belong to one of the categories in $\bigcup_{k=1}^{n} g(c_k)$. To prove e_{mid} is the only element in $\bigcup_{k=1}^{n} g(c_k)$ that has this property, let's assume e_{base} rolls up to $e_{other} \neq e_{mid}$ and e_{other} belongs to $\bigcup_{k=1}^{n} g(c_k)$ and rolls up to some element in c'. Because of the empty data movement set of categories $\{c_1, ..., c_n\}$ and c, e_{other} should also belong to $\{c_1, ..., c_n\}$ and roll up to some element in c which is impossible since c is summarizable from $\{c_1, ..., c_n\}$. Therefore, c' is summarizable from $\bigcup_{k=1}^{n} g(c_k)$. □

Example 12. Consider the non-strict dimension of Figure 8a. Although this dimension is inconsistent, category `C4` is summarizable from category `C1`. Figure 8b shows a MSR for this inconsistent dimension. As we expect, since the overall data movement set for this new repair is empty, the summarizability property between categories `C4` and `C1` is preserved. On the other hand, Figure 8c shows a SR with non-empty data movement set. Here, category `C41` is not summarizable from `C11` since element `e4` has to be considered in the summarization for parent element `e7`. Therefore, the summarizability property between categories `C4` and `C1` is lost. □



(a) An inconsistent dimension

(b) MSR preserving all summarizability properties

(c) SR losing summarizability between `C1` and `C4`

Fig. 8: Structural repairs and non-strictness

A.2 The MDX Query Language

MultiDimensional Expressions (MDX) [17] is a declarative query language for multidimensional databases designed by Microsoft. Results of MDX queries come back to client programs as data structures that need to be further processed to look like a spreadsheet. This is similar to how SQL works with relational databases. An MDX query consists of axis specifications and a cube specification. It may also contain an optional slicer specification that is used to define the slice of the cube to be viewed. A simple MDX statement has the following form:

```
SELECT <axis specification> [, <axis specification>, ...]
FROM <cube_specification>
WHERE <slicer_specification>
```

We can break this query down into pieces:

1. The `SELECT` clause defines the axes for the MDX query structure by identifying data elements to include on each axis. MDX has several ways to do this. The basic possibilities are to identify data elements either by choosing members of a category or by specifying their parent element.

2. The `FROM` clause names the cube from which the data is being queried. This is similar to the `FROM` clause of SQL that specifies the tables from which data is being queried
3. The `WHERE` clause provides a place to specify members for other cube dimensions that do not appear in the axes and filters data based on the conditions that are provided. The use of `WHERE` is optional. It is used when measures are not queried in axis specifications (For simplicity, in the remaining sections we will skip the `WHERE` clause and include measures in our axis specifications).

Example 13. We can write a simple MDX query that returns dollar and unit sales for quarters of 2010 as follows:

```
SELECT {[Measures].[Dollar Sales],
        [Measures].[Unit Sales]}
   ON COLUMNS,
       {[Time].[2010].CHILDREN}
   ON ROWS
FROM [Sales]
```

	Dollar Sales	Unit Sales
Q1,2010	1,213,380.0	2,725
Q2,2010	855,600.0	2,100
Q3,2010	1,160,419.0	2,518
Q4,2010	1,638,560.0	3,122

As stated earlier, Reusing cube views is one the important features of multidimensional databases. The following query (Lets call it $CV^{c_1,...,c_n}$) shows a template for writing cube views in MDX (for simplicity, some features of MDX that are irrelevant to our discussions have been dropped in this template):

```
SELECT [Measures].MEMBERS ON AXIS(0),
    {[c1].MEMBERS | [c1].[e1].CHILDREN} ON AXIS(1),
            ⋮
    {[cn].MEMBERS | [cn].[en].CHILDREN} ON AXIS(n)
FROM [base_cube]
```

Here, $c_1, ..., c_n$ are categories (possibly from different dimensions), `[ck].MEMBERS` retrieves all elements that belong to category c_k (`[Measures].MEMBERS` retrieves fact measures), `[ck].[ek].CHILDREN` retrieves children of element e_k from category c_k, and `base_cube` is a cube generated by selecting base categories of different dimensions (i.e. the fact table).

The following query template can be used to derive cube view $CV^{c'_1,...,c'_n}$ from another pre-computed cube view $CV^{c_1,...,c_n}$ in MDX:

```
SELECT [Measures].MEMBERS ON AXIS(0),
    {[c'1].MEMBERS | [c'1].[e1].CHILDREN} ON AXIS(1),
            ⋮
    {[c'n].MEMBERS | [c'n].[en].CHILDREN} ON AXIS(n)
FROM [CV^c1,...,cn]
```

where $c_1, ..., c_n, c'_1, ..., c'_n \in C^U$ are categories of $\mathcal{S}$ and every c'_i is an ancestor of c_i (this is called a *dependence relation* between $\langle c_1, ..., c_n \rangle$ and $\langle c'_1, ..., c'_n \rangle$ [12]).