# Logical Multidimensional Database Design for Ragged and Unbalanced Aggregation Hierarchies

Tapio Niemi
Department of Computer and Information Sciences, University of Tampere
FIN-33014 University of Tampere, Finland
tapio@cs.uta.fi

Jyrki Nummenmaa
Department of Computer and Information Sciences, University of Tampere
FIN-33014 University of Tampere, Finland
jyrki@cs.uta.fi

Peter Thanisch
Department of Computer Science, University of Edinburgh
Edinburgh, EH9 3JZ, Scotland
pt@dcs.ed.ac.uk

## Abstract

Research on logical design of OLAP cubes has tended to assume that the rollup hierarchy in a cube dimension takes the form of a balanced tree. However, experience from industry indicates that a much broader class of rollup hierarchies is of interest in practice. For example, the hierarchy tree might be unbalanced or ragged (or both), or indeed the hierarchy might not be a tree at all, but a more general acyclic directed graph structure such as a lattice. We demonstrate how dependency information can assist in the design of aggregation hierarchies. In addition to dependencies familiar from relational database theory, we use new classes of dependencies to extend logical design principles so that they include this more general notion of cube dimension hierarchies.

## 1 Introduction

Hierarchical dimensions are an essential part of On-Line Analytical Processing (OLAP). The hierarchical structure allows the user to study facts in different levels of details. Good hierarchical structures ensure correct and efficient calculation of aggregation functions: consistent and coherent structure of hierarchies decreases logical errors while efficiency of calculation is increased, for example, by decreasing redundancy in rollup paths. Logical modelling methods of these hierarchies differ from the methods used in database design. Thus, dependencies used in database design are not enough for modelling OLAP hierarchies. In this work our aim is to study which kinds of dependencies would be needed to get aggregation hierarchies more desirable for OLAP.

Much of the research on logical design for OLAP cubes has concentrated on the class of aggregation hierarchies that arise from star and snowflake schemata. In such aggregation hierarchies:

(1) the attribute labelling a column in a dimension table will correspond to a level in the aggregation hierarchy of the corresponding cube dimension, and
(2) data values occurring in that column of the dimension table will become the members of the corresponding level.

OLAP practitioners need a much more general notion of aggregation hierarchies than this. For example, an aggregation hierarchy in one of the cube's dimensions might be built from a hierarchical relationship that is modelled in one of the database tables. Example 1 illustrates this.

**Example 1** Consider a management hierarchy in an Employee table in which each row contains the information pertaining to a particular employee and one column in the table contains the employee number (attribute EmpID) and another contains the number of that employee's manager (attribute MgrID). In the OLAP cube, we have Salary as the measure and an Employee dimension. The aggregation hierarchy in the Employee dimension is the management hierarchy modelled by the EmpID and MgrID columns. This aggregation hierarchy allows us to query the total salary bill for all employees below any given manager.

We note that in Example 1, we do not have levels corresponding to *attributes*. Rather, levels correspond to *data* values occurring in the table. Thus two quite

**Proceedings of the International Workshop on Design and Management of Data Warehouses (DMDW'2001)**
Interlaken, Switzerland, June 4, 2001
(D. Theodoratos, J. Hammer, M. Jeusfeld, M. Staudt, eds.)

http://sunsite.informatik.rwth-aachen.de/Publications/CEUR-WS/Vol-39/

distinct kinds of aggregation hierarchies can arise in practice and we refer to these as *“attribute hierarchies”* and *“data hierarchies”*.

With regard to the logical design of cubes, much research has been published on the role of dependencies, such as functional dependencies, on the design of attribute hierarchies. In the present paper, we demonstrate that dependency theory is also relevant to cube design for data hierarchies. In addition to the functional dependencies, we give three new dependency classes for data hierarchies. The *anti-closure dependency* prevents calculating redundant aggregation values by disallowing “shortcuts” in hierarchies, while *non-raggedness* and *balance dependencies* ensure complete aggregations on each level in a hierarchy.

Correct summarizability is the most essential design goal for dimension hierarchies in OLAP. There are three necessary conditions for summarizability and these conditions are also assumed to be sufficient [LeSh97]:

1. disjointness of category attributes,
2. completeness, and
3. correct use of measure (summary) attributes with statistical functions.

Disjointness requires that attributes in dimensions form disjoint subsets over the elements. Completeness means that all elements exist and every element is assigned to some category on the level above it in the hierarchy. Correct use of measure attributes with statistical functions is the most complex of these requirements since it depends on the type of the attribute and the type of the statistical function. Without any prior knowledge about the semantics of the attributes, we can verify the first two conditions by using information on dependencies among attributes in a dimension.

We have organised the rest of the paper as follows. Next a review of the related work is given in Section 2. After that we present the basis of hierarchy types in Section 3. In Section 4 the new dependencies are studied. Finally, the conclusions are presented in Section 5.

## 2 Related Work

Pedersen and Jensen [PeJe99] discuss requirements for multidimensional data models. Five of their nine requirements are for dimension hierarchies:

1) explicit hierarchies in dimensions,
2) support for multiple hierarchies in a dimension,
3) correct aggregation of data,
4) support for non-strict hierarchies, and
5) data with different levels of granularity should be allowed.

Explicit hierarchies in dimensions means that hierarchies should be explicitly shown in the schema to help user's understanding of hierarchical structures in dimensions. By multiple hierarchies in a dimension, the authors mean that a dimension hierarchy is allowed to be an acyclic graph, that is, there can be several paths from the top level to the most detailed level. For example, the time dimension can have hierarchies like day-week, and day-month-year. The third item, correct aggregation of data, ensures that aggregated values are not misleading. Good structure for dimension hierarchies is essential in guaranteeing this. The idea of the fourth item, support for non-strict hierarchies, means that many to many relationships between different levels in a dimension are allowed. Finally, the last item, data with different levels of granularity should be allowed, means that it should be possible to store data in different levels of hierarchy, not only to the lowest level.

Jagadish et al. [JLS99] argue that balanced and tree structured hierarchies are not enough for many real situations but more flexible structures are needed. Zurek and Sinnwell [ZuSi99] list some non-standard requirements for data warehouses. Especially, realignment of a data warehouse schema is needed quite often since dimension hierarchies can change even regularly.

Hurtado et al. [HMV99] study how data cubes can be maintained under dimension updates. They have noticed that dimensions are not so static than it is often assumed, thus efficient maintaining of dimensions can remarkably increase the efficiency of the system. A formal model for dimension updates is presented. The model covers both domain updates and structural updates of dimensions. An algorithm to maintain a relational stored data cube under dimension updates is also presented.

Lehner et al. [LAW98] define the dimensional normal form, which is a normal form for OLAP hierarchies. A dimension is in dimensional normal form if there is a functional dependency from each lower level to the level above it. In many real situations, however, the dimensional normal form may be too strict. The authors also study a problem related to attributes that are not applicable for all instances, for example, all products do not have a colour.

Khardon et al. [KMR99] have studied Boolean dependencies in a context of example based reasoning. Boolean dependencies are a generalisation to the functional dependencies [Cod72]. While the functional dependency has a form $X \rightarrow Y$, where X and Y are sets of attributes, the Boolean dependencies allow us to use Boolean operators between attributes. Boolean dependencies can be extended from simple equality comparison of values to compare whether the values belong to the same equivalence class. This makes it possible to define constraints between value groups, e.g. age $\rightarrow$ age group.

Wijsen et al. [WNC99] study generalising temporal dependencies for non-temporal dimensions. They generalise temporal functional dependencies into *rollup dependencies* and illustrate their usability in conceptual modelling and data mining. The rollup dependencies can be used in modelling generalisation hierarchies if hierarchical levels are ordered by the finer than relation. The rollup dependencies ensure comparing equality of values in a specific level. In this way, they are a generalisation of the functional dependencies. As an example, consider a relation on hotel room reservations having attributes (ROOM_NUMBER, DATE, PRICE). An example of a rollup dependency in this relation is ROOM_NUMBER $\text{DATE}^{\text{WEEK}} \rightarrow$ PRICE, meaning that the price of the room cannot change during a week. An axiomatisation for rollup dependencies is also presented and negation with the rollup dependencies is studied.

## 3 OLAP Hierarchies

In this section we first formalise our model for the OLAP cube and then study different kinds of data hierarchies and their properties. Our formalisation is based on relational database theory and we assume that the reader knows the basics of it.

### 3.1 OLAP Model

To be able to operate with dependencies, we must formalise our notion of the OLAP cube. Generally, we assume that the cube consists of dimensions (measures can also be handled as dimensions). We use a relational approach [Cod70] that is close to the often-used star schema. However, we do not locate dimension hierarchies and the fact table in different relations but assume that the fact table also contains the dimension hierarchies. It is important to notice that this is only an abstract model used as a base of the logical design and totally independent of the implementation.

**Definition 1** The *OLAP cube* is a relation over the relation schema $H = D_1 \cup D_2 \cup \ldots D_n$, where each $D_k$ is a set of attributes and it is called a dimension schema.

If D is a dimension schema, then a relation d over D is called a *dimension*. The hierarchy representing the dimension is called the *dimension hierarchy*. The nodes in the dimension schema are called *attribute hierarchy levels*. The tuples in the dimension relation are *members* of the attribute hierarchy levels. Instead, in data hierarchies the levels and members of the levels are implied by the actual data.

In what follows we mostly use the following example of a dimension.

**Example 2** We have a dimension of sales employees. The schema has three attributes:

| | |
|---|---|
| E: | Employee ID |
| M: | Manager ID |
| G: | Job grade |

Each tuple, t, in a relation over {E, M, G} contains details about an employee, namely t[E], the employee's Employee ID, t[G], his or her Job grade, and t[M], the Employee ID of his or her manager.

### 3.2 Data Hierarchies

In the attribute hierarchy, the levels correspond to attributes and the members of the level have the names of the values that occur in the corresponding attribute's column. In a *data hierarchy* we have a parent-child relationship between just two columns, for example Employee ID and Manager ID. In addition to this, a data hierarchy can have additional attributes to indicate the level of the members, for example Job Grade for employees. This level does not necessarily correspond with the actual level of the member in the parent-child hierarchy.

Hierarchies can be classified according to their generality. Figure 1 presents a hierarchy of different data hierarchies. The most general is the acyclic digraph and the strictest is the balanced and non-ragged tree.

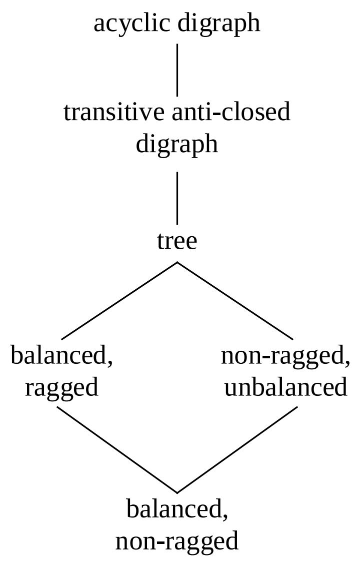

Figure 1: Hierarchy of dimension hierarchies

The transitive anti-closed graph is the complement of the transitive closure graph. The transitive closure can be constructed as follows. If there is a path from node A to node B of length 2 or more, then *add* an edge from A to B. Instead, the anti-closure can be produced as follows. If there is a path from node A to node B of length 2 or more, then *remove* an edge from A to B.

These balance and raggedness properties require that the level code is attached to each node. A graph is ragged if each node does not have a level code corresponding to its actual level in the hierarchy, whereas a graph is balanced if all leaf nodes have the same level code.

As Figure 1 indicates, we have six different kinds of hierarchies but two of them have similar properties related to OLAP hierarchies:

(1) **Acyclic.** This is the most general class. It allows the day-month-quarter-year; day-year structure, where the day-year could be called a "direct shortcut". A direct shortcut means redundant aggregation paths.
(2) **Transitive anti-closed digraph.** This is an acyclic graph with no direct shortcuts. No redundant aggregation paths are possible.
(3) **Tree.** Unique aggregation paths are guaranteed.

(4) **Unbalanced but non-ragged or balanced but ragged.** The available levels of aggregation are not equal.
(5) **Balanced and non-ragged.** Equal levels of aggregation are available.

In the next section, we study dependencies that guarantee particular hierarchies and show examples about the hierarchies.

## 4 Dependencies for Hierarchies

In this section we study relational dependencies in the context of OLAP hierarchies. We mainly concentrate on dependencies for data hierarchies. We may consider two types of dependencies:

(1) dependencies that must always hold because they say something about the semantics of the data, and
(2) dependencies that might hold in the database at the point that we want to construct the parent/child aggregation hierarchy, but which do not always hold.

For instance, it may be that at some moment each employee has a unique manager. This might be because there is a dependency, which always holds, or, alternatively, it just happens to be so at that moment but generally it is possible that an employee has several managers.

We note that, in practice, a number of other constraints on the data are needed. For example, it may be that the data hierarchy should have a unique root element for aggregation purposes. We ignore these considerations in the present paper

### 4.1 Data Hierarchy

For data hierarchies, we need the inclusion dependencies, but only in a simplified form where only a single relation is considered.

**Definition 2** Let r be a relation over R and X, Y $\subseteq$ R. There is an *inclusion dependency* [X] $\subseteq$ [Y] in r, if $\pi_X(r) \subseteq \pi_Y(r)$.

For the data hierarchy we first define a path in a relation.

**Definition 3** Let r be a relation over R and X, Y $\subseteq$ R. A sequence of tuples $t_0, t_1, ..., t_n \in r$ forms a *path* from $t_0[X]$ to $t_n[Y]$ with length n, if $t_0[Y] = t_1[X]$, $t_1[Y] = t_2[X]$, ..., $t_{n-1}[Y] = t_n[X]$.

We write $<t_0, t_1, ..., t_n>$ for a path from the tuple $t_0$ to the tuple $t_n$. A relation may contain null values but a null value cannot exist in a valid path. If the designer is considering the use of some attributes as a data hierarchy, the conditions described in the following definition must be met.

**Definition 4** A relation h over the relation schema H(XY) forms a *data hierarchy* if [Y] $\subseteq$ [X] and for each t, t' $\in$ h if there exists a path from t[X] to t'[Y], then a path from t'[Y] to t[X] does not exist.

For simplicity we assume that the left part of the hierarchy cannot contain null values. We use the example data presented in Table 1 to illustrate different classes of hierarchies.

Table 1: An example relation on management hierarchy

| empID | mgrID | grade |
|---|---|---|
| 1 | NULL | A |
| 2 | 1 | B |
| 3 | 1 | B |
| 4 | 1 | C |
| 5 | 2 | C |
| 5 | 1 | C |

The graph representing the relation of Table 1 is given in Figure 2. We use the relation of Table 1 as a running example. Typically, stricter hierarchies imply deletion of some rows from the relation.

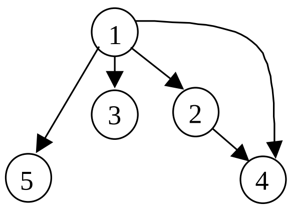

Figure 2: Directed acyclic graph representing the relation in Table 1

### 4.2 Anti-Closure Dependency

Unbalanced hierarchies often exist in applications and the vendors of OLAP products have started support them. For example, in the latest release of Microsoft Analysis Services [Mic00], it is possible to define an "unbalanced" tree. This allows us to have an EMPLOYEES relation which includes the columns EmpID and MgrID, where (as usual) the MgrID entry in a row identifies the employee's manager. Suppose that all employees are Salesmen or their managers. We want to aggregate sales by manager, by aggregating the sales of all salesmen below a manager. This is tricky because the leaf nodes can have different depths. For example, in Table 1 (see also Figure 2) leaf nodes ‘3’ and ‘4’ have different levels in the tree.

In a different context, Gottlob et al. defined the *closure dependency* [Got89]:

**Definition 5** Let r be a relation over R and X, Y $\subseteq$ R. The relation r obeys the *closure dependency* X@Y if for each pair of tuples t and t’ in r it holds: if t[Y] = t’[X], then there exists a tuple t’’ such that t’’[X] = t[X] and t’’[Y]=t’[Y].

With regard to OLAP hierarchies, the complement of the closure dependency, which we refer to as the *anti-closure dependency*, is of relevance.

We need the anti-closure dependency in order to prevent a hierarchy in which a rollup path “by-passes” intermediate nodes. The transitive anti-closure of the hierarchy in Figure 2 is given in Figure 3. There are two rollup paths from ‘4’ to ‘1’ in Figure 2. We want to eliminate the direct path, so that rollup never by-passes ‘2’.

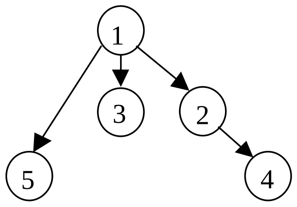

Figure 3: A transitive anti-closed graph

For the hierarchy of Example 2, we want to ensure that the transitive anti-closure of the graph, i.e. if t[M] = t’[E], there should not be some longer “path” from t’ to t using E and M values to connect tuples. This can be done by the anti-closure dependency. It guarantees that the corresponding hierarchy graph is transitively anti-closed, but this does not disallow the possibility that a node can have multiple parents, i.e. an employee reports to more than one manager.

**Definition 6** Let r be a relation over R and X, Y $\subseteq$ R. The relation r obeys the *anti-closure dependency* (*ACD*) X#>Y, if for any path <$t_0$, $t_1$, ..., $t_n$>, n $\geq$ 2, there does not exist a direct path <$v_1$, $v_m$> in r such that $t_1$ [X] = $v_1$[X] and $t_n$[Y] = $v_m$[Y].

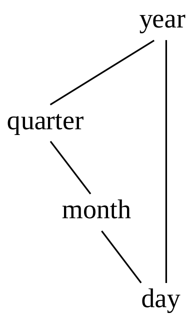

Figure 4: A dimension schema hierarchy violating ACD

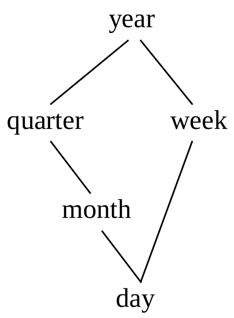

Figure 5: A dimension schema hierarchy that obeys ACD

Multiple paths between nodes are allowed but if there is a longer path between two nodes, the direct path is not allowed. Or, the same can be said in the opposite way: if there is a direct path between two nodes, the longer path cannot exist. The hierarchy in Figure 4 does not obey the ACD while the hierarchy in Figure 5 obeys it.

**Theorem 1** Let r be a relation over R and X, Y, Z, W $\subseteq$ R. If X #> Y, then XZ #> YW.

**Proof.** Let t, t’$\in$r. Since X #> Y, there is not a direct and indirect path from t[X] to t’[Y]. Adding more attributes cannot increase the number of matching values, thus there cannot exist more paths from t[XZ] to t’[YW] than from t[X] to t’[Y].

The anti-closure dependency is not transitive. A relation presented in Table 2 obeys dependencies A#>B and B#>C but not A#>C.

Table 2: Anti-closure is not transitive

| A | B | C |
|---|---|---|
| 1 | 4 | 2 |
| 2 | 5 | 3 |
| 1 | 6 | 3 |

### 4.3 Functional Dependencies

Although more general rollup hierarchies are useful, often stricter hierarchies, such as trees, are needed. The well-known functional dependencies [Cod72] can be used to force a hierarchy to a tree. In Figure 3 there is also a functional dependency between nodes. The functional dependencies can be used in both attribute and data hierarchies.

In data hierarchies, we may want to restrict our attention to Parent/Child dimensions in which each node in the dimension has just one parent. In Example 2, we can enforce this rule with the functional dependency E $\rightarrow$ M. The functional dependency E $\rightarrow$ G must also hold in our example.

**Definition 7** A relation r over R obeys a *functional dependency* $X \rightarrow Y$, if for each pair of tuples $t, t' \in r$, whenever $t[X] = t'[X]$, then $t[Y] = t'[Y]$.

OLAP hierarchies often contain non-applicable NULLs. With regard to functional dependencies, suppose two of the columns in the relation are ProductID (which will be used as a member in the cube) and Colour (which will be incorporated as a member property). ProductId → Colour holds. We assume that two products are given different ProductIds if their colours differ, even if they are identical in every other respect. However, it could be that some products do not have a colour, so in a relation, the entry in the Colour column for such products will be NULL. The functional dependency means that two rows that agree on ProductId cannot disagree on Colour. In particular, if one such row has a non-NULL entry in the Colour column, the other row must have an identical entry.

The functional dependency ProductId → Colour must hold before we can consider including the Colour column as a member property of ProductId. If we want to add Colour as a level above the ProductId level, we would need to include a special member in the Colour level for products that have NULL entries in the Colour column.

**Lemma 1** Let r be a relation over R and $X, Y \subseteq R$. If the functional dependency $X \rightarrow Y$ holds in r, then for all two tuples $t, t' \in r$ there is at most one path between $t[X]$ and $t'[Y]$.

**Proof.** (By contradiction) Assume there exist two paths from $t[X]$ to $t'[Y]$. Then there must exist an X value related to two different Y values, that is, the functional dependency does not hold.

**Theorem 2** Let r be a relation over R and $X, Y \subseteq R$. If $X \rightarrow Y$, then X #> Y.

**Proof.** Let $t, t' \in r$. According to Lemma 1 there can be at most one path between $t[X]$ and $t'[Y]$. Thus, X#>Y holds.

### 4.4 Boolean Dependencies

The following example illustrates the use of the Boolean dependencies [KRM99]. Consider the Geography dimension: Country – State – City – Neighbourhood. With ragged hierarchies, we allow NULLs in the column corresponding to potentially-missing parents (e.g., the State column for the row which has "Washington DC" in its City column). But we could insist that if we allow such raggedness, there must be a "partial" functional dependency in force which means that we must restrict the numbers of cities called "Washington DC" to one per country. There are several different cities in the USA called "Springfield", but that is acceptable, because each Springfield is located in a different State. If there were two "Washington DC"s, it would not be a problem, so long as at least one is in a State. But if neither Washington DC is in a State, then we have ambiguity. To forbid this a subtler dependency than a conventional FD is needed as Example 3 shows.

**Example 3** We have the following rule:

```
If two rows agree on City and Neighbourhood then
    If State is NULL in both rows then
        They must disagree on Country
    else
        Either One has a NULL in State
        Or they are both non-NULL in State and the State
        entries are different.
```

The Boolean dependency corresponding to the example would be: City AND Neighbourhood → State OR Country. Next we give a more formal definition for Boolean dependencies.

**Definition 8** Let R be a relation schema. The following expressions are *valid Boolean dependencies* in R:

1. A, if $A \in R$,
1. !A, if A is a valid expression,
2. (A), if A is a valid expression, and
3. A o B, if A and B are valid expressions and $o \in \{AND, OR, XOR, \rightarrow, \leftrightarrow\}$.

**Definition 9** Let r be a relation over R. The relation r obeys a *Boolean dependency* X, if r has only one tuple or for each pair of distinct tuples $t_1$ and $t_2$ in r obeys the dependency X' that is constructed from X by replacing each term A in X to $t_1[A] = t_2[A]$.

In Definition 9, we assume for the negation that $!(t_1[A] = t_2[A]) \Leftrightarrow t_1[A] \neq t_2[A]$. The tuples $t_1$ and $t_2$ have to be distinct because of problems with negation: the dependency A → !B would be never valid if $t_1 = t_2$. Although a Boolean dependency can be an arbitrary Boolean expression, we restrict our study to 'implication' dependencies. Example 4 illustrates such a Boolean dependency.

**Example 4** Table 3 obeys a Boolean dependency A→B OR C, but it does not obey A→B OR A→C.

Table 3: A→B OR C

| A | B | C |
|---|---|---|
| $a_1$ | $b_1$ | $c_1$ |
| $a_1$ | $b_1$ | $c_2$ |
| $a_1$ | $b_2$ | $c_2$ |

### 4.5 Enforcing Non-Raggedness

In our example non-raggedness means that all employees reporting to the same manager have the same job grade. We use the terms of graph theory. If $t[X, Y] \in h$, then $t[X]$ is called an underling of $t[Y]$.

**Definition 10** Let h be a data hierarchy over a relation schema H and $X, Y \subseteq H$. The set of direct underlings of $t[Y]$ is *Underlings*$(t) = \{t' \in h \mid t'[Y] = t[X]\}$.

**Definition 11** Let r be a relation over R and X, Y, W $\subseteq$ R. The relation r obeys a *non-raggedness dependency* (*NRD*) XY $\rightarrow^{NRD}$ W if $\pi_{(XY)}$r is a data hierarchy and for all tuples t $\in$ r, for all tuples t', t" $\in$ Underlings(t), t'[W] = t"[W].

In the definition, the role of W is to be a level identifier. There can be two types of hierarchies according to the values of the level identifier: 1) strict and 2) monotonic. Strict means that the child must have a different and strictly lower level than the parent, whereas monotonic means that the child must not have a higher level value than its parent has. Moreover, we say that a level identifier is *consistent* if there is a one-to-one mapping between the values of the level identifier and the actual levels of the nodes. Consistent level identifiers and non-raggedness dependency guarantee that no levels can be by-passed in rollup paths.

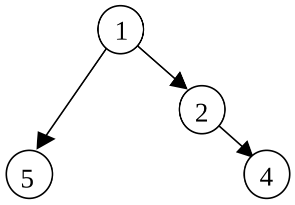

Figure 6: Ragged but balanced tree

Figure 6 shows that the completeness condition of aggregations can easily be violated in ragged hierarchies. For example, if we summarize the members of the second level, we lost the value of node '5', because it does not rollup to any member on the second level. However, the non-raggedness with consistent level identifiers is not a sufficient condition for complete aggregations. For example, the hierarchy in Figure 7 is non-ragged but unbalanced, i.e. the leaf nodes are not on the same level. Now, if we summarize the members of the lowest level, we will lose the member '3'.

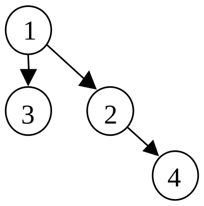

Figure 7: Non-ragged but unbalanced tree

### 4.6 Enforcing Balance

In a balanced hierarchy, all employees who are at the leaves of the organisation hierarchy have the same job grade. A hierarchy in Figure 6 is balanced, since all leaf nodes, '5' and '4' are on the same level. An example of an unbalanced tree can be seen in Figure 7. We define the set of the leaf nodes of a hierarchy as follows:

**Definition 12** Let h be a data hierarchy H. The set of the leaf nodes of h is *Leaves*(h) = {t $\in$ h | Underlings(t) is empty}.

**Definition 13** Let r be a relation over R and X, Y, W $\subseteq$ R. The relation r obeys a *balance dependency* (*BD*) XY$\rightarrow^{BD}$ W if $\pi_{(XY)}$r is a data hierarchy and for all tuples t, t'$\in$ Leaves(h), t[W] = t'[W].

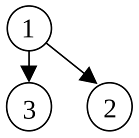

Figure 8: A non-ragged and balanced tree

Non-raggedness with consistent level identifiers and balance guarantee that aggregations are complete on every level in the hierarchy. In the balanced hierarchy all leaf nodes are on the same level, and in the non-ragged hierarchy with consistent level identifiers there must exist a node on every hierarchy level in each rollup path. An example about non-ragged and balanced hierarchy can be seen in Figure 8.

## 5 Conclusions

We have studied different aggregation hierarchies and dependencies that hold in a particular hierarchy. We have also classified the hierarchies: the most general is the acyclic digraph and the strictest is the balanced and non-ragged tree. The structure of the hierarchy has significant effect on correctness of aggregations and the storage space needed to store pre-calculated aggregation values. In general it can be said that a stricter hierarchy makes it easier to get correct aggregations. When a data hierarchy is balanced and non-ragged with consistent level identifiers, aggregations are complete on every level in the hierarchy. On the other hand, a hierarchy, which is too strict, reduces the number of possible rollup paths and thus decreases the expressive power of the OLAP cube.

## 6 References

[Cod70] Codd, E. F.: A relational model for large shared data banks, *Communications of the ACM*, **13**(6), 1970.

[Cod72] Codd, E. F.: Further normalization of the data base relational model, Data Base Systems, *Courant Computer Science Symposia Series 6*, R. Rustin (ed.), Prentice-Hall, New Jersey, 1972.

[Got89] Gottlob, G., Schrefl, M., and Stumptner, M.: On the interaction between transitive closure and functional dependencies, *MFDBS 89, the 2nd Symposium on Mathematical Fundamentals of Database*

*Systems*, J. Demetrovics et al. (eds), LNCS, Vol. 364, Springer, 1989.

[HMV99] Hurtado, A., Mendelson, A., and Vaisman A.: Maintaining data cubes under dimension updates, *Proceedings of the 15th International Conference on Data Engineering*, IEEE Computer Society Press, 1999.

[JLS99] Jagadish, H., Lakshmanan, L., and Srivastava D.: What can hierarchies do for data warehouses?, *Proceedings of the 25th International Conference on Very Large Data Bases*, M. Atkinson et al. (eds), Morgan Kaufmann, 1999.

[KMR99] Khardon, R., Mannila, H., Roth, D.: Reasoning with examples: propositional formulae and database dependencies, *Acta Informatica*, **36**(4): 267-286, 1999.

[LAW98] Lehner, W., Albrecht, J., and Wedekind, H.: Normal forms for multidimensional databases, *Proceedings of the 10th International Conference on Scientific and Statistical Data Management (SSDBM'98)*, 1998.

[LeSh97] Lenz, H.-J. and Shoshani, A.: Summarizability in OLAP and statistical data bases, *Ninth International Conference On Scientific And Statistical Database Management (SSDBM)*, 1997.

[Mic00] Jacobson, R.: *Microsoft SQL Server 2000 Analysis Services Step By Step*, Microsoft Press, 2000.

[PeJe99] Pedersen, T. and Jensen, C.: Multidimensional data modeling for complex data, *Proceedings of the International Conference on Data Engineering (ICDE'99)*, 1999.

[WNC99] Wijsen, J., Ng, R., and Calders, T.: Discovering roll-up dependencies, *The Fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD 1999*, ACM, 1999.

[ZuSi99] Zurek, T. and Sinnwell, M.: Data warehousing has more colours than just black & white, *VLDB'99, Proceedings of 25th International Conference on Very Large Data Bases*, M. Atkinson et al. (eds.), Morgan Kaufmann, 1999.