Aggregate Queries in Peer-to-Peer OLAP

Mauricio Minuto Espil
Pontificia Universidad Catolica Argentina
mminuto@uca.edu.ar

Alejandro A. Vaisman
Universidad de Buenos Aires
avaisman@dc.uba.ar

ABSTRACT

A peer-to-peer (P2P) data management system consists essentially of a network of peer systems, each maintaining full autonomy over its own data resources. Data exchange between peers occurs when one of them, in the role of a *local* peer, needs data available in other nodes, denoted *the acquaintances* of the local peer. No global schema is assumed to exist for any data under this computing paradigm. Henceforth, data provided by an acquaintance of a local peer must be adapted, in a manner that answers to queries posed by local peer users conform the view those users have of their data. Because a multidimensional database normally consists in a collection of views of aggregated data, a careful translation process is needed in this case, in order to transform any summary concept that appears in a peer acquaintance into a summary concept meaningful to the requesting peer. We present a model for multidimensional data distributed in a P2P network, and a query rewriting technique, that allows a local peer to propagate OLAP queries among its acquaintances, obtaining a meaningful and correct answer.

Categories and Subject Descriptors: H.2.4 [Systems]: Distributed Databases, Query Processing

General Terms: Algorthms, Design, Theory.

Keywords: Data Warehousing, OLAP, P2P Computing.

1. MOTIVATION

Let us suppose a common situation nowadays, where a company analyzes benefits and risks of acquiring shares of other targeted companies. Before any decision can be made, the investing company needs to have a good knowledge about the critical factors of success and failure of all the targeted companies, while adapting its goals to these factors. An integrated decision-making information system that reflects the state of business of the investor company that includes the state of business of all the targeted companies, is needed therefore. However, in this scenario, designing an additional centralized warehouse clearly would not be the best solution. Among other reasons, shares usually change hands often enough for a centralized design to be achieved on time, and, in addition, because any company can invest in shares of any other, the role of investor may result interchangeable. Companies would rather continue operating in an autonomous way therefore and provide information for decision-making one to each other when needed, in a flexible, cooperative way. An architecture for cooperative interchange of decision-making information seems to be a natural solution for this problem. Of course, this setting could be extended to a more general scenario, where organizations interchange summarized information. For example, public offices at different levels of government, at the state, province or country level, need to exchange summarized information about, for instance, tax paying. A new cooperative, distributed OLAP paradigm will fit properly this necessity. Here, information in each node (a company in our former example) will be organized in *dimensions*, *fact tables* and *cubes* as usual, all designed according to its own needs, and the system will operate in an autonomous fashion, without the intervention of any central data warehouse. This assumption implies that some dimensions modeled in one node may not be present in some others, and the hierarchies of common dimensions, at the schema and/or instance level, may differ one from each other. Data integration must be thus performed, to reflect how each node views the information of the others. In this work we propose a Peer-to-Peer (P2P) data model for addressing this problem, assuming that we are working within the boundaries of an organization.

Throughout the paper we will be using the following example. There are two nodes storing information about two companies, shareholders one of each other. Each node stores information about their dimensions of interest in dimension tables, conforming a hierarchy of levels [1, 9] and one or more fact tables recording events from the company. Node 1 holds information about products (in dimension *Product*), and geographic organization (in dimension *Geography*). There is a base fact table *Sales* with attributes *itemsold*, and *city*, and a measure *amnt*, representing gross sales. Node 2 holds information about lease-holds (in dimension *Lease-hold*), customers (in dimension *Customer*), and geographic organization (in dimension *Geography*). There is a

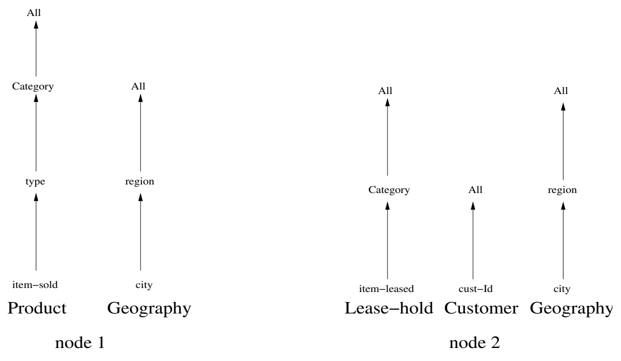

Figure 1: Dimension Schemas at Node 1, and Node 2

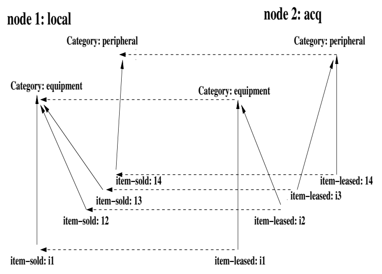

Figure 2: The "equipment" mapping problem

base fact table *Leases* with attributes *item-leased*, *cust-Id*, and *city*, and measure *amnt*. The hierarchies for dimensions in node 1 and node 2, are depicted in Figure 1.

The Problem

Sales and leases are, in our example, similar transactions that involve economic goods and produce income to a company. We can give them the common generic name of *trades*. Integrating information on trades implies more than a common name. It also supposes a common view of the dimensions that describe sales with the dimensions that describe leases. In Figure 1, this is the case of dimensions *Product* in node 1, and *Lease-hold* in node 2, that represent information on the items subject of each type of trade, and dimensions *Geography* in both nodes that represent information on location. It is not the case of dimension *Customer* in node 2, however, because there is no counterpart for it in node 1. At the instance level, a unified view of members of the integrated dimensions must be obtained; members of one dimension must be mapped to members of the other, and conversely. Unfortunately, this mapping is not always trivial. Dimension hierarchies may differ, due to different classification criteria (like dimensions *Product* and *Lease-hold* in Figure 1). A more involved situation may appear, as Figure 2 shows. Here, the concept "equipment" is denoted on level *category* of dimension *Product* in node 1 and on level *category* of dimension *Lease-hold* in node 2. However, the extension of both categories differ: for example, the first one includes an item as a child in the hierarchy (i_3) that the other one misses.

Let us suppose that a user in node 1 poses the query "Give me the total amount in trades by product category and region". According to the structure of the data warehouse described above, nodes 1 and 2 may contribute with information on the amount of their sales and leases respectively. Nonetheless, while the information on sales and dimensions stored in node 1 are understood for the user posing the query, the information stored in node 2 does not necessarily conform the nomenclature expected by this user, who is not supposed to know (at least completely) the meaning of concepts used there. For instance, members of dimension *lease-Hold* on level *category* in node 2 must be interpreted in the query result as if they were members of dimension *Product* on level *category* on node 1. Thus, a mapping mechanism for achieving this interpretation is needed. As the "equipment" class problem shows, a mapping may not suffice. Moreover, we must attend to missing levels. For instance, what if the query asks for trades grouped by product type instead of product category?. There is no level with similar meaning to that of level *Product-type* in dimension *Lease-hold*. At least two different semantics can be adopted for the query: (a) we can return the result considering only the nodes where the requested levels appear in the dimension schema (in this case node 1 only), clearly the simpler, although incomplete solution; or (b) we can try to *infer* the missing values for the absent levels whenever possible.

Our proposal

We propose a model for the problem stated above, considering each node as a *peer* in a cooperative query system, and allowing each peer a high degree of autonomy. Each node involved in the system defines a *context* where it becomes the *local* peer, and all its dimensions and fact tables are considered local henceforth. The rest of the nodes that connect with the local peer, are considered *acquaintances* of the local peer within this context. A *LAV* (Local As View) integration approach is used, but it does not rely on mapping exclusively. We propose a systematic *revise and map* strategy for defining how an instance of a dimension in an acquaintance is viewed from the local peer. Operators for revising dimension instances, in the spirit of those defined in Hurtado *et al* [10], are used to *reclassify* members in order to redefine the extension of conflicting class members (as "equipment" and "peripheral" in node 2). A set of *redefinition rules* is generated upon the operators, and a revision of the instance with those rules is made, creating a separated *revised* instance [16]. Redefinition rules and mappings are defined in the local peer, and propagated to the acquaintances within a *mapping acquisition phase*, and when activated, they generate *mapping tables* and *revised rollup functions* in the acquaintances.

We also introduce the class of P2P-OLAP queries. Informally, a P2P-OLAP query over this data model, is a query involving aggregate functions, defined over generalizations of fact tables and dimensions that are distributed in a P2P network. We characterize P2P OLAP queries as a set of datalog rules with aggregation. Based on this characterization, we discuss an evaluation procedure for these queries. Whenever a query is submitted to a node, that node fixes a context where all its fact tables and rollup functions become local. The posed query is then rewritten for propagation to any of the acquaintances, introducing references to the appropriate mapping tables and revised rollup functions. Thus, the query result depends on the node where the query is posed and may change as changes in the revision and mapping process occur. Moreover, if the mapping is incomplete, our rewriting technique employs a *bottom-up homomorphism preserving completion* approach that allows the system to always produce a query result.

The remainder of this work is organized as follows: in Section 2 we discuss related work. In Section 3 we present the basic multidimensional model. Section 4 introduces the notion of the multidimensional P2P data model. In Section 5 we present a P2P OLAP queries, and define a method for query evaluation in this environment, based on query rewriting. We conclude in Section 6.

2. RELATED WORK

Several models for integrating data distributed in a P2P environment have been proposed. While Gribble *et al* introduced the problem of data management in P2P networks [6], Serafini *et al* [17] were the first ones to propose a logical framework for maintaining consistency under updates in a P2P network. In [11] the authors present a model based on first order logic, for querying general P2P databases, introducing explicitly the notion of domain mappings. The same notion, but in this case in the form of *mapping tables* is discussed in [12]. The use of variables instead of constants in the domain is studied there. Halevy *et al* [7] and Tatarinov *et al* [18] present a general framework based on mappings, introducing the concept of *certain answers* of queries. Complexity of query answering is also studied. In [5], the role of first order logic for P2P data management system is discussed. The authors claim that P2P data management systems cannot be modeled after standard classical logic. The role of mappings is confined, under their approach, to data migration only. Results concerning answering datalog queries is studied in this context. Calvanese *et al* [3] present a general framework based in epistemic logic, extending the classical framework, in order to deal with inconsistencies at the conceptual level. In [15], a rule model for integration based on graphs is presented. This model is strongly inspired in the LAV and GAV paradigms discussed in [14]. All of these works deal with data integration in a general context. Nevertheless, the intractability of query answering in the general case forces to study the problem in more constrained settings. To the best of our knowledge, the only work we know approaching integration in a multidimensional context is the one by Cabibbo and Torlone [2], but confined exclusively to define conditions for integration of dimensions on data marts. Our query model is is based on datalog with aggregation, thus producing tractable results when acyclic topologies are involved, and admits local views of all data in the network. The main difference with the approaches discussed above consists in the use of a *revise and map* technique. This technique, based on the notion of belief revision, allows producing concrete redefinitions of concepts in the acquaintances, not mere views, triggered by the local peer, in order to ease query answering.

3. DIMENSIONS AND FACT TABLES REVISITED

In OLAP systems, aggregated data in the form of fact tables is presented to analysts in a multidimensional way. Facts are aggregated according to the dimensions of interest. Dimensions are therefore structured in levels of granularity and a dimension instance is conformed by a collection of parent-child relationships among level members called roll-ups. For practical reasons, dimension instances are usually presented in the form of tables. For the sake of simplicity, we will consider in our model homogeneous dimensions only [13, 8]. The same concepts can be extended to the treatment of heterogeneous dimensions.

3.1 Dimensions

Let us consider the following sorts and their associated finite non-empty sets: a sort $\mathbf{D}$ of dimension names, a sort $\mathbf{L}$ of level names, and a sort $\mathbf{C}$ of coordinate values. Each dimension name d in $\mathbf{D}$ is associated with a set of levels in $\mathbf{L}$ by a relation $\_ : \_ : \mathbf{D} \times \mathbf{L}$. We will denote $d : l$ the pair (d, l) being a member of relation $\_ : \_$. Each level l in $\mathbf{L}$ is associated with a set of coordinate values $Iset(d : l) \subseteq \mathbf{C}$, denoted the *instance set* of level l in d. A coordinate value c is a *member* of level $d : l$; thus, the expression $d : l : c$ indicates that c is a member of $Iset(d : l)$. The following conditions hold:

- there is a distinguished level $All \in \mathbf{L}$;
- level $All \in d$, for each dimension $d \in \mathbf{D}$, ;
- $Iset(d : All) = \{all\}, \forall d \in \mathbf{D}$;
- For each pair (d, l) such that $d : l$ holds, $l \neq All$ if and only if $all \notin Iset(d : l)$;
- $Iset(d_1 : l_1) \cap Iset(d_1 : l_2) = \phi$, for all pairs $(d_1, l_1) \neq (d_1, l_2)$, such that $d_1 : l_1, d_1 : l_2$ holds, and $l_1, l_2 \neq All$.

Definition 1 (Hierarchy). *A* hierarchy *for L is a structure $(L, \preceq_L)$ where $L \subseteq \mathbf{L}$ is a non-empty finite set of level names containing the distinguished level name All, and $\preceq_L$ is a binary relation on L such that:*

- *$\preceq_L^*$, the transitive and reflexive closure of $\preceq_L$, is a partial reflexive order;*
- *there exists one level name l_{inf_L} in L such that for every level name $l \in L$, $l_{inf_L} \preceq_L^* l$; we call level l_{inf_L}, the bottom level of $\preceq_L$;*
- *$l \preceq_L^* All$ holds for every level name l in L;*
- *if l_a and l_b are level names in L, and $l_a \preceq_L l_b$, then there not exist level names l_1,..., l_k in L such that $l_a \preceq_L l_1 \preceq_L \ldots \preceq_L l_k \preceq_L l_b$;*
- *if l_a is a level name in L, and $l_a \neq All$, then there exists some level name l_b in L such that $l_a \preceq_L l_b$;*

In what follows $\preceq_L^+$ will stand for the non- reflexive transitive closure of $\preceq_L$.

DEFINITION 2 (DIMENSION SCHEMA). *A* dimension schema *is a named structure $dname = (L, \preceq_L)$ where:*

- *$dname \in \mathbf{D}$ is the name of the dimension.*
- *$L \subseteq \mathbf{L}$ is a finite set of level names, which contains the distinguished level name All.*
- *$(L, \preceq_L)$ is a hierarchy for L;*

EXAMPLE 1. *Let us consider the dimension* Geography *at node 1 in our running example. A schema for dimension* Geography *is the structure Geography $= (G, \preceq_G)$, such that $G = \{city, region, All\}$, and $\preceq_G = \{(city, region), (region, All)\}$.*

DEFINITION 3 (DIMENSION INSTANCE). *Let $d = (L, \preceq_L)$ be a dimension schema; $Iset(d : l)$ the instance set of level l of dimension d, for each level $l \in L$. A dimension instance for d is a set of tuples with schema L, obtained substituting each level $l \in L$ by a member in $Iset(d : l)$, and satisfies the functional dependencies $l \rightarrow l'$, where $(l, l') \in \preceq_L$.*

DEFINITION 4 (ROLLUP FUNCTION). *Let $d = (L, \preceq_L)$ be a dimension schema, I an instance of d, and l_1 and l_2 levels in L such that $l_1 \preceq_L^+ l_2$ or $l_1 = l_2$ A* rollup *function $rup_{l_1 \rightarrow l_2}^d$ with signature $\mathbf{C} \rightarrow \mathbf{C}$, is the set $\{ t.l_1 \mapsto t.l_2 \mid t \in I \}$, a function that results from projecting tuples in the instance I of dimension d over the ordered pair of levels (l_1, l_2). We call a member of a rollup function a* rollup.

DEFINITION 5 (FACT TABLE). *Let us now define the sort $\mathbf{F}$, standing for fact tables. A* fact table schema *is a structure $f = (sp, ms)$, where $f \in \mathbf{F}$ is a fact table, sp is a tuple $(d_1, ..., d_t)$, where d_j, $j = 1, ..., t$, are different dimensions in $\mathbf{D}$, called the* space *for f, and ms is the name of a set of values, called the* measure *of the fact table. Moreover, given a fact table schema $f = (sp, ms)$, a* cell *for f is a tuple resulting from mapping each bottom level in d_j that occurs in sp to a value in $Iset(d_j : lbottom_j)$. Given a fact table schema $f = (sp, ms)$, a* fact table instance *over it is a partial function which maps cells for f to values in set ms.*

4. A P2P MULTIDIMENSIONAL MODEL

In this section we present a model for multidimensional databases operating in a Peer-to-Peer manner, where each node holds only the dimensions of interest at that node, and the structure of the fact tables is composed by the bottom levels of the local dimensions. Each fact table is populated locally, and queries are evaluated globally. We will base the definition of the model in the environment presented in Section 3.

4.1 Dimension Peers

DEFINITION 6 (SET OF DIMENSION PEERS). *A set containing dimensions d_1,...,d_r representing the same semantic concept d is called a* set of dimension peers, *denoted generically $\mathcal{P}_d$. Each dimension $d_i \in \mathcal{P}_d$ is called* a dimension peer *with respect to any other dimension $d_j \in \mathcal{P}_d$.*

There is at most one dimension peer in one node. The role of locality depends only on the node where a query is posed, denoted a *context.* Hence, a context defines one node among the peers that is considered local. The notion of context is useful to represent nodes in a network. In this paper we do not consider the problems related to the network and its topology.

Notation. We denote $\mathbf{N}$ a set of names of nodes in a network; $\mathcal{R}$ in $\mathbf{D} \times \mathbf{N}$, is a relation such that each tuple (d, p) in $\mathcal{R}$ means that a dimension d is assigned to node p. Given a dimension d_i in a set $\mathcal{P}_d$, and a node $n \in \mathbf{N}$, a (partial) function $Peers$ with signature $\mathbf{D} \times \mathbf{N} \rightarrow \mathbf{D}$ produces the dimension $d_j \in \mathcal{P}_d$ located in node n that is a peer of d_i. A copy of this function must be present in every node.

Because the schema of the acquaintance dimension may differ from the schema of the local dimension, mapping members of the acquaintance dimension to members in the local dimension requires the definition of a correspondence between levels in both dimensions. This correspondence is materialized with a function $Levels_{d_{acq}}^{d_{local}}$, where d_{acq} stands for the acquaintance dimension, and d_{local} stands for the local dimension. We are only interested in correspondences that preserve the order between levels; otherwise, we would be able to roll-up members with higher granularity to members with lower granularity.

DEFINITION 7 (LEVEL CORRESPONDENCE). *Given a pair of dimensions d_i and d_j, both members of $\mathcal{P}_d$, with schemas $d_i = (L_i, \preceq_{L_i})$ and $d_j = (L_j, \preceq_{L_j})$ respectively, a* level correspondence *from d_i to d_j is an injective partial function $Levels_{d_i}^{d_j}$, with signature $\mathbf{L} \rightarrow \mathbf{L}$, that maps levels in L_i onto levels in L_j. An* order preserving level correspondence *between d_i and d_j is a level correspondence between d_i and d_j, satisfying that, for each pair of levels l_1 and l_2 in dimension d_i such that (a) $l_1 \preceq_{L_i}^+ l_2$, and (b) l_1 and l_2 are members of $dom(Levels_{d_i}^{d_j})$, $Levels_{d_i}^{d_j}(l_1) \preceq_{L_j}^+ Levels_{d_i}^{d_j}(l_2)$ holds. Moreover, $Levels_{d_i}^{d_j}$ (All) = All.*

Notice that the definition above does not imply a complete correspondence. Some levels in the acquaintance dimension may not have corresponding levels in the local dimension. Also, the correspondence given by function $Levels_{d_i}^{d_j}$ may be different from the correspondence given by function $Levels_{d_j}^{d_i}$. The first one is used when dimension d_j is the local dimension. The second one is used when dimension d_i is the local dimension. Copies of both level correspondence functions must be stored in both nodes.

4.2 Mappings

Once the correspondence among levels has been defined, it is necessary to define mappings among members. The following definition adapts the concept of mapping tables [11] to the OLAP setting.

DEFINITION 8 (BASE MAPPING TABLES). *Let d_{local} be a peer dimension in some set of dimension peers $\mathcal{P}_d$, local in some context and d_{acq} be a dimension peer in $\mathcal{P}_d$, different from d_{local}. Let l_{local} and l_{acq} be levels in d_{local} and d_{acq} respectively, such that $Levels_{d_{acq}}^{d_{local}}(l_{acq}) = l_{local}$. A partial*

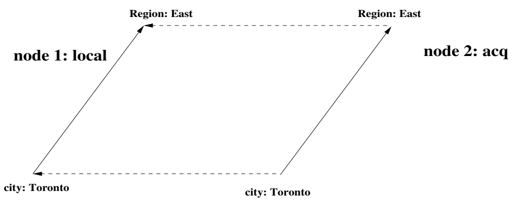

Figure 3: Homomorphism between dimensions *Geography* in nodes 1 and 2

function $map_{l_{acq} \to l_{local}}^{d_{acq} \to d_{local}}$*, with signature* $\mathbf{C} \to \mathbf{C}$*, assigning members in level* l_{local} *of dimension* d_{local} *to members in level* l_{acq} *of dimension* d_{acq}*, is denoted a* base mapping table*. If function* $map_{l_{acq} \to l_{local}}^{d_{acq} \to d_{local}}$ *is total the base mapping table is said a* complete *mapping.*

In an ideal situation, the union of all defined base mapping tables should operate as an homomorphism between the dimension instances regarded as structures. Let us describe this situation formally.

Definition 9 (Consistent Mapping). *Let a function* $Levels_{d_{acq}}^{d_{local}}$ *be an order preserving level correspondence. A base mapping table* $map_{l_{acq} \to l_{local}}^{d_{acq} \to d_{local}}$ *is* consistent *if and only if the following holds:*
For each member m such that:

$$map_{l_{acq} \to l_{local}}^{d_{acq} \to d_{local}}(m) \text{ is defined,}$$

if there exists some member m' *satisfying that:*

$$rup_{l'_{acq} \to l_{acq}}^{d_{acq}}(m') = m$$

for some level $l'_{acq} \in dom(Levels_{d_{acq}}^{d_{local}})$*, then*

$$rup_{l'_{local} \to l_{local}}^{d_{local}}(map_{l'_{acq} \to l'_{local}}^{d_{acq} \to d_{local}}(m')) = map_{l_{acq} \to l_{local}}^{d_{acq} \to d_{local}}(m),$$

where $l'_{local} = Levels_{d_{acq}}^{d_{local}}(l'_{acq})$.

This (ideal) situation is depicted in Figure 3. Ideal situations, however, are not likely to occur in real cases (except when mapping bottom levels). Figure 4 shows a case where defining a mapping that implies an homomorphism is not possible, referring to our running example. City "Ottawa" in dimension *Geography* at node 2 rolls up to region "east". When the dimension becomes an acquaintance of dimension *Geography* at node 1, region "east" at node 2 cannot be mapped, because "Ottawa" rolls up to region "north east" at node 1.

Definition 10 (Conflicting Members). *Let a table* $map_{l_{acq} \to l_{local}}^{d_{acq} \to d_{local}}$ *be a consistent mapping table. We call a member* $m \in dom(map_{l_{acq} \to l_{local}}^{d_{acq} \to d_{local}})$ *a* mappable *member. If a member m is not mappable we call it a* conflicting member.

We will present a *revise and map* approach for solving the problem introduced by conflicting members.

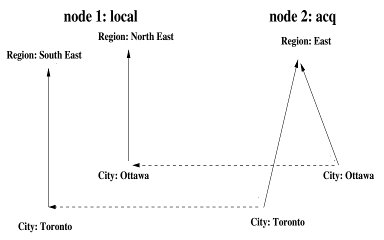

Figure 4: Impossibility of Mapping Definition

4.3 Revise and Map Approach

A *revise and map* approach (RAM), consists in revising the instance of the acquaintance dimension, adapting it to the meaning of members of the local dimension, and then mapping members, using the revised instance. A revision process of an acquaintance dimension d is accomplished by the systematic application of revision operators to the instance I_d of d, specified by a user at the local node. In our setting, a user can apply two operators for revising a dimension: *revise by reclassifying* and *revise by splitting*. These operators, applied to the original instance I_d, produce a new instance rev_I_d containing a set of revised tuples. Then, the union of all unmodified tuples in the instance I_d with the modified tuples in instance rev_I_d is used for deriving a set of *revised rollup functions*, denoted *rev_rup*. These functions are stored in the acquaintance node and will be later used for evaluation of queries submitted from the local node. The revision operators *revise by reclassifying* and *revise by splitting* are based on the update operators reclassify and split, as described in [10]. Succinctly, update operators like reclassify and split, modify a given instance of a rollup function by replacing a set of its pairs (old rollups) with another set of new pairs (new rollups). We slightly change the semantics of these operators. In our approach, we see dimension instances as derived from conclusions of a proof system. In this setting, an old rollup $l_s : a \mapsto l_t : b$ constitutes a default rule of aggregation, stating that all facts that *aggregate* on member a of level l_s aggregate on member b of level l_t. We derive a *redefinition rule* from each new rollup specified in the revision operation, and redefinition rules operate as exceptions to defaults, when concluding where facts aggregate. From a theoretical perspective, a prioritized default theory is built, adding all redefinition rules derived from the specified revision operations to the set of rules obtained from old rollups in the original dimension. A priority policy that prefers redefinition rules over ordinary roll-ups rules is defined as a component of the theory, in the form of an ordering relation among rules, and the set of aggregations inferred from this theory defines the *revised* dimension instance. An efficient algorithm exists for computing a dimension instance, revised with redefinition rules. The reader is referred to [16] for a detailed treatment.

The problem with members "equipment" in dimensions "Product" and "Lease-hold" discussed in section 1 can be easily solved with a RAM approach. In the first case, we can reclassify the items in dimension *Lease-hold* such that "equipment" means the same thing for both dimensions. Both revised fragments of those dimensions are shown in

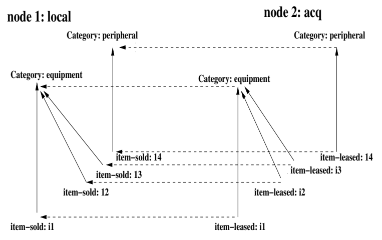

Figure 5: Revised instances of dimensions *Product*, and *Lease-hold*

Figure 5. Once the revised dimension instance without conflicting members is available, consistent and complete base mapping tables can be defined, and complete and consistent answers for queries can be produced. A RAM specification may be submitted by a local node user at any time, independently of query submission, and activates the firing of a set of processes that we denote a *mapping acquisition phase*. A mapping acquisition phase always forces a check of consistency for every mapping produced, with respect to the resulting revised instance. It only admits the definition of consistent mappings. Operational details of a mapping acquisition phase are beyond the scope of this paper.

4.3.1 Handling Conflicting Members

The RAM technique described above, allows eliminating potential conflicting members, producing consistent mappings. Nevertheless, because query submission is independent of the activation of mapping acquisition phases, mapping tables may be incomplete when a query is posed. A result for the query should be produced anyway. Therefore, we need a method for producing query answers even when information in mapping tables is incomplete.

Our method is based on the notion of homomorphism described earlier: if a member m' on level l'_{acq} of dimension d_{acq} is not consistently mapped to any member in level l_{local} of dimension d_{local}, we can drill-down m' and consider the set $Drill = \{m | m' = rup^{d_{acq}}_{l_{acq} \to l'_{acq}}(m), mappable(m)\}$, for a level $l_{acq} \preceq^{+}_{L} l'_{acq}$, l_{acq} the closest to l'_{acq} down in the hierarchy. Because each mappable member $m \in Drill$ has a consistent mapping defined, we can compute at query time a set of the form $rup^{d_{local}}_{l_{local} \to l'_{local}}(map^{d_{acq} \to d_{local}}_{l_{acq} \to l_{local}}(m))$, that is, the set of members of dimension d_{local} on level l_{local} that the mappable members m (children of m') happen to map. This procedure is repeated, varying level l_{acq}, down in the hierarchy, while there exist descendants of member m' in level l_{acq} that are not mappable, ending when all descendants are mappable or, there is no level left for consideration. In this case, there exists a path in the members hierarchy that reaches member m' with no mappable descendants of m'; any answer for a query involving level l'_{acq} will be incomplete therefore.

In our approach, the process described above occurs at the mapping acquisition phase. We maintain two tables $non\_conflict^{d_{acq}}_{l'_j}$ and $conflict\_rup^{d_{acq}}_{\to l'_j}$, for each level l'_j in d_{ack}. The unary relation $non\_conflict^{d_{acq}}_{l'_j}$ contains all mappable members on level l'_j in dimension d_{acq}. The ternary relation $conflict\_rup^{d_{acq}}_{\to l'_j}$ contains a tuple (l'_i, m_i, m_j), for all pairs $(l'_i : m_i, l'_j : m_j)$ such that:

- m_i and m_j are members of levels l'_i and l'_j, respectively, in dimension d_{acq};
- member m_j does not have a mapping defined onto the local dimension;
- member m_i is the last consistently mapped member in the revised instance of dimension d_{acq}, that is a descendant of member m_j;

4.4 Facts

Let us consider again the example presented in Section 1. Facts tables (in our example, *Sales* and *Leases*) have a meaning in the nodes where they are stored. In a global setting, they are meaningless, unless we abstract them as members of some common class, like trades in the example. Let us define the sort **G**, standing for generic fact tables. Let us suppose in our example, that a generic fact table *Trades* is a member of **G**. As an ordinary fact table, a generic fact table f has a schema $f = (space, measure)$.

DEFINITION 11 (FACT TABLE PEERS). *A set* $\{f_1, \ldots, f_r\}$ *of fact tables, all with a same measure and all corresponding to the same generic fact table* $f \in \mathbf{G}$*, is called a* set of fact table peers. *A fact table* $f_i \in \{f_1, \ldots, f_r\}$ *is called* a fact table peer *that specializes* f*. The set of all dimensions* d_k*,* $k = 1, ..., n$*, in the space for* f*, must be included in the set of all dimensions in the space for every fact table peer that specializes* f*. As in the case of dimensions, the role of local, and acquaintance of the local can be assigned to fact tables peers, depending on a given context. In our running example,* Sales *and* Leases *are both fact table peers specializing the generic fact table* Trades.

Notation. $\mathcal{F}$ in $\mathbf{F} \times \mathbf{N}$, is a relation such that each tuple (f, n) in $\mathcal{F}$ means that a fact table f is assigned to node n. Given a fact table f_i in a set $\{f_1, \ldots, f_r\}$ of fact table peers, we have a (partial) function $FtoG$ with signature $\mathbf{F} \to \mathbf{G}$ that maps a fact table f_i to its generic fact table f. Given a generic fact table f, and a node $n \in \mathbf{N}$, we have a (partial) function $GtoF$ with signature $\mathbf{G} \times \mathbf{N} \to \mathbf{F}$ that returns the fact table peer f_i in node n that is generalized by generic fact table f.

When a query is posed to a node, a context is set. Within this context the fact table peer that belongs to the node and specializes the generic fact table appearing in the query is considered the local fact table. In order to propagate the query to any other node, an acquaintance fact table in such node must be determined, if it exists. Functions $FtoG$ and $GtoF$ provide this information. Thus, a copy of both functions must be present in both nodes. The maintenance of functions $FtoG$ and $GtoF$ is accomplished at a phase that we call *publish-subscribe generalization phase*. As in the case of the mapping acquisition phase, we do not consider here details on generalization phases.

5. QUERIES

Let $\mathcal{N} = \{n_1, ..., n_r\}$ be a subset of $\mathbf{N}$, and there is a context where node $n_l \in \mathcal{N}$ is local.

A *P2P-OLAP query* q, over a (multidimensional) space $(d_1, \ldots, d_n)$, posed over node n_l is an expression of the form:

$$\begin{aligned} q(Z_1, \ldots, Z_n, aggr(M), \mathcal{N}) \leftarrow\ & Fact(X_1, \ldots, X_n, M), \\ & rup^{d_1}_{lbottom_1 \to l_{t_1}}(X_1, Z_1), \\ & \ldots, \\ & rup^{d_n}_{lbottom_n \to l_{t_n}}(X_n, Z_n); \end{aligned}$$

where:

- $d_i, i = 1, ..., n$ are dimensions in node n_l, different one from each other, with schemas $d_i = (L_i, \preceq_{L_i})$, $lbottom_i \in L_i$, the bottom level of dimension d_i, and $l_{t_i} \in L_i$,
- $Fact \in \mathbf{G}$ is a generic fact table such that there exists a fact table $baseFact = GtoF(Fact, n_l)$ with schema $Fact = (space, measure)$, and $space = (d_1, ..., d_n)$,
- $aggr$ is a distributive aggregate function, compatible with $measure$,
- $rup^{d_i}_{lbottom_i \to l_{t_i}}, i = 1, ..., n$, are roll-up functions.

EXAMPLE 2. *The expression:*

$$\begin{aligned} q(Z_1, Z_2, SUM(M), \{n_1, n_2\}) \leftarrow\ & \\ & Trades(X_1, X_2, M), \\ & rup^{Geography}_{city \to region}(X_1, Z_1), \\ & rup^{Product}_{item-sold \to category}(X_2, Z_2); \end{aligned}$$

*is a P2P-OLAP query over the multidimensional space (*Geography*,* Product*). Query q asks for the total amount of* trades *by* region *and* category.

In what follows, we will present the semantics for P2P-OLAP queries, based on a rewriting technique.

5.1 Query Rewriting

Fact tables and dimensions in a P2P-OLAP query q are meant to comprehend all their peers distributed among the nodes in set $\mathcal{N}$. We must rewrite q as a query computable in the local node, that integrates the result of a set of queries, computable each on a distinct node p, that refer to fact tables and dimensions distributed along the network. Some syntax and some semantics must be chosen for the queries; we choose those of non-recursive Datalog with aggregation as defined in [4]. In what follows, we call generically $d'_1, ..., d'_n$ the dimensions in node p that are peers of dimensions $d_1, ..., d_n$ in the schema of $Fact$.
Before any rewriting can be accomplished, we must determine whether a node p is relevant for query q or is not.

DEFINITION 12 (RELEVANT NODE). *A node $p \in \mathcal{N}$ is* relevant *for a P2P-OLAP query q if and only if:*

- *there exists a fact table $baseFact_p$ in p such that:*

$$baseFact_p = GtoF(Fact, p),$$

- *for each dimension d_k, $k = 1, ..., n$, in the schema of $Fact$, there exists a level l_{s_k} in d_k such that a pair (l'_{s_k}, l_{s_k}) exists in $Levels^{d_k}_{d'_k}$, d'_k is the peer dimension of d_k in node p, and level l_{s_k} precedes or is equal to level l_{t_k} in query q.*

We rewrite query q as the following datalog program (we use capital letters, eventually subscripted, as variables):

$$\begin{aligned} q(Z_1, \ldots, Z_n, Ag(\{..., M_j, ...\})) \quad \leftarrow \quad & \ldots, \\ & q^p(Z_1, \ldots, Z_n, M_j), \\ & \ldots; \qquad (1) \end{aligned}$$

where a query q^p, for any node p relevant to query q, is defined as:

$$\begin{aligned} q^p(Z_1, \ldots, Z_n, Ag(M)) \quad \leftarrow \quad & q_p(B_1, Y_1, \ldots, B_n, Y_n, M), \\ & rup^{d_1}_{B_1 \to l_{t_1}}(Y_1, Z_1), \\ & \ldots, \\ & rup^{d_n}_{B_n \to l_{t_n}}(Y_n, Z_n); \qquad (2) \end{aligned}$$

where $Ag = aggr$, except in the case of $aggr = COUNT$ where $Ag = SUM$, B_k, $k = 1, ..., n$, are level variables, and atoms $rup^{d_k}_{B_k \to l_{t_k}}(Y_k, Z_k)$ are considered true for all roll-ups to level l_{t_j} in dimension d_k with origin in the level that substitutes B_k.

Queries q^p and query q, as defined in (2) and (1) respectively, can be safely evaluated in node n_l, the node where the query is posed, providing that the result of every query q_p is known and predicate q_p is considered extensional. Queries q_p constitute the distributed portions in the evaluation process of query q. Thus, each query q_p must be consistently defined for evaluation on each node p, and must refer exclusively to fact tables and dimensions available on that node.

EXAMPLE 3. *If the query in example 2 is submitted to node n_1, the following datalog program is generated, for evaluation on node n_1.*

$$\begin{aligned} q(Z_1, Z_2, SUM(\{M_1, M_2\})) \leftarrow\ & q^{n_1}(Z_1, Z_2, M_1), \\ & q^{n_2}(Z_1, Z_2, M_2); \\ q^{n_1}(Z_1, Z_2, SUM(M)) \leftarrow\ & q_{n_1}(B_1, Y_1, B_2, Y_2, M), \\ & rup^{Geography}_{B_1 \to region}(Y_1, Z_1), \\ & rup^{Product}_{B_2 \to category}(Y_2, Z_2); \\ q^{n_2}(Z_1, Z_2, SUM(M)) \leftarrow\ & q_{n_2}(B_1, Y_1, B_2, Y_2, M), \\ & rup^{Geography}_{B_1 \to region}(Y_1, Z_1), \\ & rup^{Product}_{B_2 \to category}(Y_2, Z_2); \end{aligned}$$

Here, predicates q_{n_k}, $k = 1, 2$, are considered extensional, and correspond to the results of two distributed queries, one for node n_1 and the other for node n_2.

5.1.1 Propagating Queries

We concentrate in what follows on the definition of queries q_p. Because node p is relevant for query q, a table $baseFact_p = GtoF(Fact, p)$, the fact table peer that specializes generic fact table $Fact$, must exist on every node p. Nonetheless, $baseFact_p$ may have dimensions in its schema not present in the schema of $Fact$. Those dimensions are simply suppressed from the query.

If node p is the local node n_l, we define query q_p as:

$$\begin{aligned} q_p(B_1, Y_1, \ldots, B_n, Y_n, M) \leftarrow\ & baseFact_{n_l}(Y_1, \ldots, Y_n, M), \\ & B_1 = lbottom_1, \\ & \ldots, \\ & B_n = lbottom_n; \end{aligned} \qquad (3)$$

where $baseFact = GtoF(Fact, n_l)$.

If p is an acquaintance, according to the RAM approach, references to mappings and roll-ups applied on the revised dimension instances must be introduced in the query. We thus define a query q_p for an acquaintance as:

$$\begin{aligned} q_p(B_1, Y_1, \ldots, B_n, Y_n, aggr(M)) \ \leftarrow & \\ & baseFact_p(X'_1, \ldots, X'_n, M), \\ & rev\_rup^{d'_1}_{lbottom'_1 \to l'_{s_1}}(X'_1, Y'_1), \\ & map^{d'_1 \to d_1}_{l'_{s_1} \to B_1}(Y'_1, Y_1), \\ & B_1 = l_{s_1}; \\ & \ldots, \\ & rev\_rup^{d'_n}_{lbottom'_n \to l'_{s_n}}(X'_n, Y'_n), \\ & map^{d'_n \to d_n}_{l'_{s_n} \to B_n}(Y'_n, Y_n), \\ & B_n = l_{s_n}; \end{aligned} \qquad (4)$$

where, for each $k = 1, ..., n$, B_k are level variables, level l_{s_k} is the closest level that precedes level l_{t_k} in query q, down in the hierarchy, such that the pair (l'_{s_k}, l_{s_k}) is a member of $Levels^{d_k}_{d'_k}$. Atoms $rev\_rup^{d'_k}_{lbottom'_k \to l'_{s_k}}$ are considered true on all roll-ups derived from the revised instance of dimension d'_k, as it is left by the last mapping acquisition phase, and atoms $map^{d'_k \to d_k}_{l'_{s_k} \to B_k}(Y'_k, Y_k)$, are considered true for all mappings from level l'_{s_k} in dimension d'_k to the level that substitutes variable B_k in dimension d_k, as they were defined in that phase.

Program (4) will compute, for each member m' that appears as a coordinate of a cell in the acquaintance fact table, a member m according to the path: $d'_k : lbottom'_k \mapsto d'_k : l'_{s_k} \mapsto d_k : B_k$, for some level B_k in dimension d_k. Program (2) will in turn compute the targeted member on level $d_k : l_{t_k}$ that corresponds to m.

An optimization can be introduced in (2) because of (3) and (4): if the level that substitutes variable B_k coincides with level l_{t_k} in (2), there is no need of any roll-up for dimension d_k; the roll-up reference in (2) can be omitted and atom:

$$q_p(B_1, Y_1, \ldots, B_k, Y_k, \ldots, B_n, Y_n, M),$$

replaced by atom:

$q_p(B_1, Y_1, \ldots, Z_k, \ldots, B_n, Y_n, M)$ (Variable B_k is omitted).

Example 4. *For the query in the example 3, two queries are generated and distributed: a query q_{n_1} for node n_1:*

$$\begin{aligned} q_{n_1}(B_1, Y_1, B_2, Y_2, M) \ \leftarrow\ & Sales(Y_2, Y_1, M), \\ & B_2 = item - sold, \\ & B_1 = city; \end{aligned}$$

and a query q_{n_2} for node n_2:

$$\begin{aligned} q_{n_2}(B_1, Y_1, B_2, Y_2, SUM(M)) \ \leftarrow & \\ & Leases(X'_2, V', X'_1, M), \\ & rev\_rup^{Lease-hold}_{item-leased \to category}(X'_1, Y'_1), \\ & map^{Lease-hold \to Product}_{category \to B_2}(Y'_1, Y_1), \\ & B_2 = category, \\ & rev\_rup^{Geography}_{city \to region}(X'_2, Y'_2), \\ & map^{Geography \to Geography}_{region \to B_1}(Y'_2, Y_2), \\ & B_1 = region; \end{aligned}$$

Notice the presence of variable V' standing for the absent for dimension Customer*. Notice also that an optimization can be made; the fragment:*

$$\begin{aligned} & q_{n_2}(B_1, Y_1, B_2, Y_2, M), \\ & rup^{Geography}_{B_1 \to region}(Y_1, Z_1), \\ & rup^{Product}_{B_2 \to category}(Y_2, Z_2); \end{aligned}$$

in the definition of q^{n_2} in example 3 can be replaced by:

$$q_{n_2}(Z_1, Z_2, M),$$

because $B_1 = region$ and $B_2 = category$.

5.1.2 Bottom-Up Completion

Substituting roll-up predicates and introducing mappings in the query does not suffice, because we need to produce the most complete result under incomplete mappings. The result of queries q_p would not include the aggregation of the measure of facts in the acquaintance that roll-up to a conflicting member. We use the relations $non\_conflict^{d'_k}_{l'_{s_k}}$ and $conflict\_rup^{d'_k}_{\to l'_{s_k}}$, maintained during mapping acquisition phases, for defining *expanded rollup functions* that blur the distinction of reached levels, so ensuring that the ranges of the expanded functions do not include conflicting members but their closest mappable antecessors instead. We then replace atoms $rev\_rup^{d'_n}_{lbottom'_k \to l'_{s_k}}(X'_k, Y'_k)$ in the definition of queries q_p in (4) by atoms of the form $exp - rup^{d'_k}_{lbottom'_k \to l'_{s_k}}(X'_k, B'_k, Y'_k)$, with predicates $exp - rup^{d'_k}_{lbottom'_k \to l'_{s_k}}$ defined as follows:

$$\begin{aligned} exp-rup^{d'_k}_{lbottom'_k \to l'_{s_k}} & (X'_k, B'_k, Y'_k) \leftarrow \\ & rev\_rup^{d'_k}_{lbottom'_k \to l'_{s_k}}(X'_k, Y'_k), \\ & non\_conflict^{d'_k}_{l'_{s_k}}(Y'_k) \\ & B'_k = l'_{s_k}; \\ exp-rup^{d'_k}_{lbottom'_k \to l'_{s_k}} & (X'_k, B'_k, Y'_k) \leftarrow \\ & rev\_rup^{d'_k}_{lbottom'_k \to l'_{s_k}}(X'_k, W'_k), \\ & conflict\_rup^{d'_k}_{\to l'_{s_k}}(B'_k, Y'_k, W'_k); \quad (5) \end{aligned}$$

The second argument B'_k in $exp-rup^{d'_k}_{\to l'_{s_k}}(X_k, B'_k, Z_k)$ is a level variable.

Example 5. *For dimension Geography in node n_2, we have a conflict with the region where member "Ottawa" rolls up. If we do not reclassify "Ottawa", we would need to generate the following:*

$$\begin{aligned} exp-rup^{Geography}_{city \to region} & (X'_1, B'_1, Y'_1) \leftarrow \\ & rev\_rup^{Geography}_{city \to region}(X'_1, Y'_1), \\ & non\_conflict^{Geography}_{region}(X'_1) \\ & B'_1 = region; \\ exp-rup^{Geography}_{city \to region} & (X'_1, B'_1, Y'_1) \leftarrow \\ & rev\_rup^{Geography}_{city \to region}(X'_1, W'_1), \\ & conflict\_rup^{Geography}_{\to region}(B'_1, Y'_1, W'_1); \end{aligned}$$

In this case, the pair ("Ottawa", "central") is a roll-up in function $rev\_rup^{Geography}_{city \to region}$ and $conflict\_rup^{Geography}_{\to region}$ contains the tuple (city, "Ottawa", "central"); thus $exp-rup^{Geography}_{\to region}$ contains the tuple ("Ottawa", city, "Ottawa").

A similar technique for blurring levels in the origin of mappings is needed, enforcing only mappable members in their domains. We replace all occurrences of atoms of the form $map^{d'_k \to d_k}_{l'_{s_k} \to B_k}(Y'_k, Y_k)$, in the bodies of queries q_p defined in (4), by atoms of the form $exp-map^{d'_k \to d_k}_{l'_{s_k} \to B_k}(B'_k, Y'_k, Y_k)$, that bottom-up completes the mapping in the case of conflicting members. We define predicate $exp-map^{d'_k \to d_k}_{l'_{s_k} \to B_k}$ as:

$$\begin{aligned} exp-map^{d'_k \to d_k}_{l'_{s_k} \to l_{s_k}} & (Y'_k, B_k, Y_k) \leftarrow \\ & map^{d'_k \to d_k}_{l'_{s_k} \to l_{s_k}}(Y'_k, Y_k); \\ & non\_conflict^{d'_k}_{l'_{s_k}}(Y'_k) \\ & B_k = l_{s_k}; \\ exp-map^{d'_k \to d_k}_{l'_{s_k} \to l_{s_k}} & (Y'_k, B_k, Y_k) \leftarrow \\ & conflict\_rup^{d'_k}_{\to l'_{s_k}}(B'_k, Y'_k, W'_k), \\ & map^{d'_k \to d_k}_{B'_k \to B_k}(Y'_k, Y_k); \quad (6) \end{aligned}$$

Example 6. *For dimension Lease−hold in node n_2, we have a conflict with category "equipment". If we do not reclassify leased items, we need to generate:*

$$\begin{aligned} exp-map^{Leases \to Sales}_{category \to category} & (Y'_2, B_2, Y_2) \leftarrow \\ & map^{Leases \to Sales}_{category \to category}(Y'_2, Y_2); \\ & non\_conflict^{Lease-hold}_{category}(Y'_2) \\ & B_2 = category; \\ exp-map^{Leases \to Sales}_{category \to category} & (Y'_2, B_2, Y_2) \leftarrow \\ & conflict\_rup^{Lease-hold}_{\to category}(B'_2, Y'_2, W'_2), \\ & map^{Leases \to Sales}_{B'_2 \to B_2}(Y'_2, Y_2); \end{aligned}$$

Theorem 1. *Any component m' of a tuple (m'_b, l', m') in relation $exp-rup^{d'_k}_{lbottom'_k \to l'_{s_k}}$ that results from applying the datalog program defined in* (5) *on relations $conflict\_rup^{d'_k}_{\to l'_{s_k}}$, $non\_conflict^{d'_k}_{l'_{s_k}}$, and the revised instance of dimension d_k, as generated at the mapping acquisition phase, is a mappable member.*

Proof. Let us analyze the first datalog rule of the datalog program that defines predicate $exp-rup^{d'_k}_{\to l'_{s_k}}$. The constant l'_{s_k} that equals variable B'_k in the head of the first rule indicates that any member m' that substitutes variable Y'_k is a member of level l'_{s_k}. This member m' must satisfy the formula $non\_conflict^{d'_k}_{l'_{s_k}}(m')$ in order to appear as the third component of a tuple in the result. This is the same that asserting that member m' is mappable.
In the second rule, the situation is slightly different; the second argument in the head is variable B'_k, also appearing as an argument of predicate $conflict\_rup^{d'_k}_{\to l'_{s_k}}$. According to the definition of relation $conflict\_rup^{d'_k}_{\to l'_{s_k}}$ in subsection 4.3.1, the member m' that substitutes variable Y'_k, must be a mappable member of level B'_k. $\square$

Theorem 2. *Any component m' of a tuple (m', l, m) in relation $exp-map^{d'_k \to d_k}_{l'_{s_k} \to l_{s_k}}$ that results from applying the datalog program defined in* (6) *on the extensions of predicates $conflict\_rup^{d'_k}_{\to l'_{s_k}}$, mapping tables and revised rollup functions, generated at the mapping acquisition phase, is a mappable member.*

Proof. Let us analyze the first datalog rule of the datalog program (6) that defines predicate $exp-map^{d'_k \to d_k}_{l'_{s_k} \to l_{s_k}}$. Any member m' of level l'_{s_k} that substitutes variable Y'_k must satisfy the formula $non\_conflict^{d'_k}_{l'_{s_k}}(m')$ in order to appear in the second argument of any tuple satisfying the head. This is the same that asserting that member m' is mappable.

In the second rule, the first argument in the head is variable B'_k, also appearing as an argument for predicate $conflict\_rup^{d'_k}_{\to l'_{s_k}}$. In this case, the substitution of variable B'_k by a level l' varies on a level by level basis; it depends on the path reaching the conflicting member that substitutes variable Y'_k.

Any member m' that substitutes variable Y'_k is a member of level l', and, by definition of relation $conflict\_rup^{d'_k}_{\rightarrow l'_{s_k}}$ in subsection 4.3.1, m' is mappable. □

Theorems 1 and 2 ensures that, for every mappable bottom member $lbottom'_k$ in the acquaintance dimension d'_k there is a path to some member in a level B_k in the local dimension d_k, such that level B_k precedes or is equal to level l_{t_k} in the query. Because roll-up functions are complete (we have assumed homogeneity), there is always a path from member $lbottom'_k$ to some member in level l_{t_k}, that is to a coordinate for a cell in the query result. This configures the best attempt to complete a query result when mappings are not complete.

5.1.3 Complexity

The following theorem shows a bound for the time complexity of our query rewriting method.

THEOREM 3. *The time complexity of the method for query rewriting presented above is proportional to* $O(\eta, \delta, \lambda)$, *where* $\eta = |(\mathcal{N})|$, δ *is the maximal number of dimensions a fact table peer has in its schema,* λ *is the maximal number of levels in the schema of any dimension peer.*

PROOF. *(Sketch)* It is easy to see that the time complexity of the method is proportional to η, because query q is expanded in queries q^p, which in turn expand to a query q_p each, for each $p \in (\mathcal{N})$. For each query q_p the method generates bodies in the defining rule, one for each dimension, thus deriving the proportionality to δ. Finally, the method searches for a level l_{s_k} in dimension d_k that is closest to level l_{t_k}, down in the hierarchy of levels of d_k. The worst case of this search entails the proportionality to λ in the complexity. □

6. CONCLUSION

We have presented a model for answering multidimensional queries in a P2P OLAP setting. In this environment, a peer receiving the query is considered *local* and the rest of the peers are considered *acquaintances* of the former. We have proposed and discussed two integration principles:

- *revise and map* (RAM) for integrating dimensions under the common view of the local peer. This process occurs in a *mapping acquisition phase*, and
- specialization of *generic fact tables* for conciliating facts, defined and maintained at a *fact generalization phase*.

Finally, a method for rewriting and propagating queries has been proposed, in order that all peers can understand P2P-OLAP queries.

Several questions have not been tackled, like: (a) the definition of a protocol for metadata interchange in both phases of integration; (b) deciding how redundant views of aggregated data are stored and maintained in the different peers. This question is extremely important when considering issues like performance and response time.

7. REFERENCES

[1] L. Cabibbo and R. Torlone. Querying multidimensional databases. In *Proceedings of the 6th DBPL International Workshop*, pages 253–269, East Park, Colorado, USA, 1997.

[2] L. Cabibbo and R. Torlone. Dimension compatibility for data mart integration. In *Proceedings of the 12th Italian Symposium on Advanced Database Systems*, pages 6–17, Cagliari, Italy, 2004.

[3] D. Calvanese, G. De Giacomo, M. Lenzerini, and R. Rosati. Logical foundations of peer-to-peer data integration. pages 241–251, Paris, France, 2004.

[4] M. Consens and A. Mendelzon. Low complexity aggregation in Graphlog and Datalog. In *Proceedings of the 3rd ICDT Conference, LNCS n.470*, pages 379–394, 1990.

[5] E. Franconi, G. Kuper, A. Lopatenko, and L. Serafini. A robust logical and computational characterisation of peer-to-peer database systems. In *Proceedings of the First DISP2P International Workshop, LNCS n.2944*, pages 64–76, Berlin,Germany, 2003.

[6] S. Gribble, A. Halevy, I. Zachary, M. Rodrig, and D. Suciu. What can peer-to-peer do for databases, and viceversa? In *Proceedings of the 4th International WebDB Workshop*, pages 31–36, Santa Barbara, California, 2001.

[7] A. Halevy, Z. Ives, D. Suciu, and T. I. Schema mediation in peer data management systems. In *Proceedings of the 19th IEEE-ICDE International Conference*, pages 505–516, Bangalore, India, 2003.

[8] C. Hurtado and A. Mendelzon. Reasoning about summarizability in heterogeneous multidimensional schemas. *Proceedings of the 8th ICDT International Conference, LNCS n.1973*, pages 375–389, 2001.

[9] C. Hurtado, A. Mendelzon, and A. Vaisman. Maintaining data cubes under dimension updates. *Proceedings of 15th IEEE-ICDE International Conference*, pages 346–355, 1999.

[10] C. Hurtado, A. Mendelzon, and A. Vaisman. Updating OLAP dimensions. *Proceedings of the 2nd DOLAP International Workshop*, pages 60–66, 1999.

[11] A. Kementsietsidis, M. Arenas, and R. Miller. Managing data mappings in the hyperion project. In *Proceedings of the 19th IEEE-ICDE International Conference*, pages 732–734, Bangalore, India.

[12] A. Kementsietsidis, M. Arenas, and R. Miller. Mapping data in peer-to-peer systems: Semantics and algorithmic issues. In *Proceedings of the ACM SIGMOD International Conference*, pages 325–336, San Diego, USA, 2003.

[13] A. Lenz, H. y Shoshani. Summarizability in olap and statistical databases. In *Proceedings of the 9th SSDBM International Conference*, pages 132–143, Washington, USA, 1997.

[14] M. Lenzerini. Data integration: a theoretical perspective. pages 233–246, Madison, Wisconsin, 2002.

[15] P. McBrien and A. Poulovassilis. Defining peer-to-peer data integration using both as view rules. In *Proceedings of the First DISP2P International Workshop, LNCS n.2944*, pages 91–107, Berlin, Germany.

[16] M. Minuto and A. Vaisman. Revising aggregation hierarchies in olap: a rule-based approach. In *Data and Knowledge Engineering 45(2)*, pages 225–256, 2003.

[17] L. Serafini, F. Giunchiglia, J. Mylopoulos, and P. Bernstein. The logical relational model: Model and proof theory. In *Technical Report 0112-23,ITC-IRST*, 2001.

[18] I. Tatarinov and A. Halevy. Efficient query reformulation in peer-data management systems. In *Proceedings of the ACM SIGMOD International Conference*, pages 539–550, Paris, France, 2004.