Using Ontologies to Discover Fact IDs

Alberto Abelló
Dept. Enginyeria de Serveis i Sistemes d'Informació
Univ. Politècnica de Catalunya
E-08034 Barcelona, Spain
aabello@essi.upc.edu

Oscar Romero
Dept. Llenguatges i Sistemes Informàtics
Universitat Politècnica de Catalunya
E-08034 Barcelona, Spain
oromero@lsi.upc.edu

ABSTRACT

Object identification is a crucial step in most information systems. Nowadays, we have many different ways to identify entities such as surrogates, keys and object identifiers. However, not all of them guarantee the entity identity. Many works have been introduced in the literature for discovering meaningful IDs, but all of them work at the logical or data level and they share some constraints inherent to the kind of approach. Addressing it at the logical level, we may miss some important data dependencies, while the cost to identify data dependencies at the data level may not be affordable. In this paper, we propose an approach for discovering fact IDs from domain ontologies. In our approach, we guide the process at the conceptual level and we introduce a set of pruning rules for improving the performance by reducing the number of ID hypotheses generated and to be verified with data. Finally, we also introduce a simulation over a case study to show the feasibility of our method.

Categories and Subject Descriptors

H.2.1 [**Database Management**]: Logical Design; H.2.7 [**Database Management**]: Database Administration—*data warehouse*

General Terms

Algorithms

Keywords

Identifiers, Reengineering, Ontologies

1. INTRODUCTION

Object identification is a crucial step in most information system engineering process. Identification mechanisms guarantee that a real world entity will be uniquely identified in the system and distinguished from the others. However, not all mechanisms guarantee the object identity [28]. The uniqueness of the surrogates (a common way to implement IDs in relational databases) is useless for data quality, since they are internal IDs. Meaningful IDs are crucial for assuring the quality of data in many reengineerning processes; e.g., for ensuring the quality when migrating, converting and merging systems, for data cleansing, data integration or for building *data warehouse* (DW) systems.

Like in most information systems, the DW design has been typically carried out manually, and the experts' knowledge and experience are crucial to identify relevant *multidimensional* (MD) knowledge contained in the sources. However, there is an important difference from other systems, since creating a DW does not require the addition of new information but rearrange that already existing (indeed, the DW is nothing else than a strategic view over the organization data). Consequently, some research efforts have proposed the automation of the DW design in order to free this task of being (completely) performed by an expert, and facilitate the whole process. In this way, in those scenarios where the analysis capabilities of the data sources are not (fully) known, it is possible to help the DW designer to identify and elicit unknown analysis capabilities. These unknown capabilities may potentially provide strategic advantages.

In star-shape schemas design, where the central fact table has not an atomic ID, but one composed by the *foreign keys* (FK) pointing to the dimension tables, having such complex IDs is a must. Specifically, DW systems design can benefit from knowledge about IDs in the following ways: (i) identifying facts (since good candidates are precisely those with composed IDs), (ii) identifying analysis dimensions for a given fact (since they are precisely the components of the IDs), (iii) generate metadata to inform the user of the different spaces where data can be placed (in case of different composed IDs for the same fact), (iv) avoid sparse cubes (since, by definition, IDs are minimal sets), and (v) find overlapping cubes (to improve the physical design by storing together those data that can be easily analyzed together).

Currently, we may find several approaches to discover *functional dependencies* (FDs or "→") and/or *candidate keys* (CKs), see Section 2. Note that the traditional ID concept is just a specific case of FD (see, for example [2]). Nevertheless, these approaches work either at the logical or data level, and they share some inherent constraints. Those working over the logical schema are tied to the design decisions made when devising the system (for example, denormalization of data) and these decisions have a big impact on the data semantics captured in the schema. These approaches make some unrealistic assumptions such as completeness

DOLAP'10, October 30, 2010, Toronto, Ontario, Canada.

of the data structures (i.e., all the constraints of the domain of interest are captured at the logical level). For this reason, most automated approaches for identifying IDs require to address this task at the instance level. However, these methods have various drawbacks: they tend to overlook composite IDs (essential when dealing with multidimensional databases), propose solutions that are computationally expensive, and register drops in performance when a large number of attributes or instances are processed.

In this paper we propose an approach to discover meaningful IDs from domain ontologies. The software engineering area has claimed to use conceptual representations of the domain on the top of systems to have an up-to-date and accurate formalization of the system domain [16]. This approach has given rise to concepts such as OBDA (Ontology-Based Data Access) [18] and nowadays, ontologies are used in many fields such as data integration, conceptual modeling as well as the semantic web. In our approach, we guide the process at the conceptual level and introduce a set of pruning rules for improving the performance by reducing the number of ID hypotheses generated and to be verified with data. Our algorithm is relevant since, despite the importance of object identification, most *description logics* (DL), the most extended ontology languages and also the basis of the OWL language (a W3C recommendation) do not provide identification mechanisms, and only very expressive DL (that are not suitable for real world applications due to their computational complexity) incorporate them [4]. For instance, the only way to specify IDs in OWL DL is by means of one-to-one binary relationships which are clearly not enough. Furthermore, the fact that most DLs do not consider n-ary relationships makes impossible to assert composite IDs in ontologies. In short, there is no way to express that $\{A,B\} \rightarrow C$ holds (where A, B and C are concepts), because in most ontology languages, roles are binary predicates relating one class to another class.

All in all, an algorithm to discover meaningful IDs benefiting from the semantics of an ontology is a relevant issue for many real world systems. To our knowledge, this is the first approach introduced in the literature proposing to lead this task at the conceptual level. As result, it is able to generate less hypotheses to be validated with data and therefore, it performs better than current approaches.

Section 2 points out some related work, and Section 3 sketches the framework where we tested our work. Section 4 establishes the foundations, and Section 5 explains the details of the algorithm. Finally, Section 6 gives some statistics of a simulation over a middle-sized ontology to show the feasibility of our method. Section 7 concludes the paper.

2. RELATED WORK

There have been some works proposing to model the DW from a conceptual formalization of the domain. However, most of them must be carried out manually since they present a step-by-step guide to be followed by an expert (see [22] for further details). Only a few of them try to automate some part of this process, but the degree of automation achieved is rather low. Specifically, these approaches consist of a detailed requirement elicitation stage (to be performed manually) and an automated analysis of the data sources. Later, both stages are put in common, conciliating in this way the data sources and requirements (for instance, see [3, 11]). In these methods the requirement elicitation stage leads the process and main design decisions are captured in this step whereas the analysis of the data sources is a posteriori. Indeed, the analysis of the data sources is just a complementary task to the requirement elicitation processes, instead of a guide for it.

On the other hand, the automation of the DW design process from relational schemas has been thoroughly addressed in the literature (e.g., [13, 17, 23] among others). All these methods rely on a thorough analysis of the relational sources. In these cases, dimensional data is discovered by design patterns based on FK and CK constraints. In MD design, it is well-known that facts and dimensions must be related by many-to-one relationships, to give rise to a meaningful MD space. In a relational schema, this kind of relationships (i.e., *mandatory* FDs) are modeled by means of CK and FK constraints. For this reason the accuracy of results got by these methods depends on the existence of CK and FK in the schema (maybe disabled by the DBA to improve performance, maybe lost due to denormalization below 3NF).

To our knowledge, this is the first work addressing this issue at the conceptual level. We may find many works for discovering FDs and CKs (e.g., [7, 8, 14, 15, 25, 30] among others), but they work at the logical or data level and they share some inherent constraints. To avoid missing some important data dependencies, these approaches make some unrealistic assumptions such as completeness of the data structures (i.e., all the constraints of the domain of interest are captured at the logical level).

Addressing this issue at the data level though, would imply such an effort that may not be worth (as explained in [13]). If the logical schema lacks semantics, we may need to sample such amount of data that for a large number of attributes (since we have to blindly test any possible combination of attributes) it would rise an unaffordable computational complexity ([27] generates an exponential number of queries). Moreover, these approaches exclusively work over instances and they cannot easily tolerate erroneous data (that may generate fake IDs that do not really hold or overlook real ones).

Unlike these approaches, we use ontological knowledge to guide the search and we do not exclusively rely on data. It has two main benefits. On one hand, we rely on a clear picture of the domain of interest free of logical/physical design decisions and we are able to considerably improve the performance by reducing drastically the number of hypotheses to be verified with data. On the other hand, this approach may be used for assuring the quality of data, as the feasible IDs found for each concept are extracted from knowledge in the domain ontology. Thus, it opens new perspectives, since we may also use this information to detect erroneous data in the database (i.e., those feasible IDs refuted by data).

3. FRAMEWORK

Most methods that automate the design process focus on identifying the MD knowledge contained in the sources regardless of the requirements. These methods are known as *supply-driven* approaches (see [29]). It is well known that these approaches suffer from one drawback: they tend to generate too many results. Consequently, they unnecessarily overwhelm users with blindly generated combinations whose meaning has not been analyzed in advance. Eventually, they put the burden of (manually) studying and se-

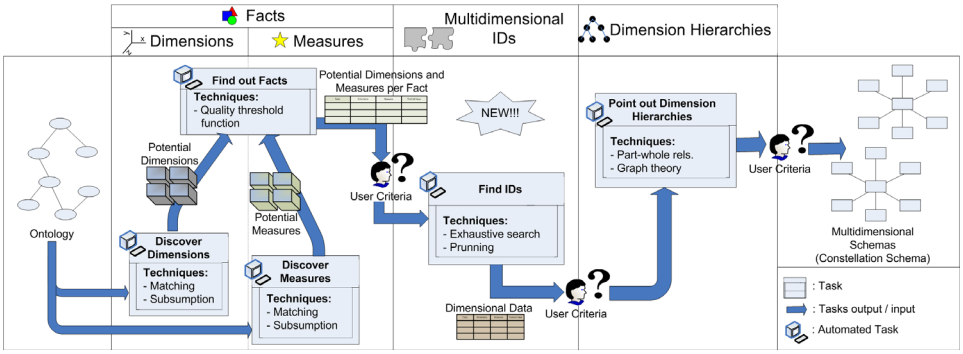

Figure 1: Method Overview

lecting results provided onto the designer's shoulder, but the time-consuming nature of this task can render it unfeasible when large data sources are considered. Filtering the results provided by these approaches is a must.

In this paper, we take a *user-centered* approach to support the end-user requirements elicitation and the DW design tasks. First, we start by fully analyzing the data sources to identify, also without considering requirements as other approaches do, the MD knowledge they capture (i.e., data likely to play a dimensional role). However, we drive this search and are able to filter the results so that we only propose promising analysis dimensions to the user. After this feedback, the user elicits the MD requirements. Finally, once requirements have been stated, we can automatically generate the DW conceptual schema (by shaping the MD knowledge extracted from the sources according to the requirements). Thus, we say it is user-centered since the automatic process produces a manageable amount of information, and the feedback of the user is still needed to select results among those obtained from analyzing the sources.

Even though this work could be integrated in other supply-driven approaches like [17], we demonstrate its feasibility by including it as a new feature in the framework (i.e. AMDO) presented in [21]. This is a supply-driven method, since it carries out an exhaustive search of potential facts among all the concepts in the domain, which may generate many results. In this sense, our approach overcomes the problem by filtering and priorizing the amount of information shown to the user at the end of every step. We claim to derive all the MD knowledge contained in the ontology, prune it according to statistical evidences, and eventually let the user select from results obtained according to his/her requirements. On the one hand, we are conciliating requirements with data available. On the other hand, we believe that it is easier to carry out the requirements elicitation from knowledge proposed by AMDO than carrying it out from scratch.

Figure 1 depicts a schematic overview of this framework. It has three well-differentiated tasks, which identify concepts likely to play MD roles and therefore, eventually produce MD schemas:

1. The first task looks for potential subjects of analysis (i.e., facts). In the literature we can find different approaches to discover facts but most of them are hardly automatable (automatically identifying facts is a hard task [17]). Thus, potential facts are ranked (and maybe filtered) according to some user-defined heuristic (independently of its position in the ranking, a fact may be of interest or not for the user and he/she has to decide it). Most methods rely on heuristics such as table cardinalities or having numerical attributes. We enrich those heuristics by adding two new variables, i.e., number of potential measures and dimensions. This task, therefore, is divided in two subtasks: (a) Discover measures and (b) Discover dimensions.
2. The second task discovers sets of concepts likely to be used as an ID for each fact choosen by the user. They are compound of those elements identified as dimensions in the previous task. In short, we look for concepts being able to univocally identify objects of analysis (i.e., factual data) and produce interesting data cubes. Similar to the previous task, IDs identified are ranked according to the sparsity of the multidimensional cube (see [11] for a precise definition and formula to calculate it) they generate for the corresponding fact. Too sparse data cubes can be filtered and not presented to the user. Eventually, the user will choose, among the IDs proposed, those of his/her interest.
3. The last task gives rise to dimension hierarchies. We aim to identify relevant aggregation paths looking for part-whole relationships among the dimensions.

Thus, in the first term, for each concept we would estimate its likeliness of being a fact (different functions can be used, for example those in [21] or [26]). Afterwards, the user should choose some of the concepts from the list. The novelty of our approach is that, similar to the quality rule for ranking facts, all the dimensional concepts related to a given fact will be analyzed to check whether they form an ID or not, and how sparse is the space these IDs induce over that fact. If the space induced by an ID is not too sparse (a *sparsity threshold* can be introduced here), it is proposed to the user for the corresponding fact (together with the analysis dimensions composing it). All previous works propose all dimensions to the user without filtering them in any way, generating most of the times too many results with a high computational cost.

4. FOUNDATIONS

Importantly, our approach discovers IDs guided by the domain knowledge captured in the ontology. First, we formally discuss the implications of the ID concept at the conceptual level. We make use of a generic conceptual notation: by *concepts* we refer to an ontology concept (classes in OWL notation) or a datatype, and by *relationships* to ontology roles

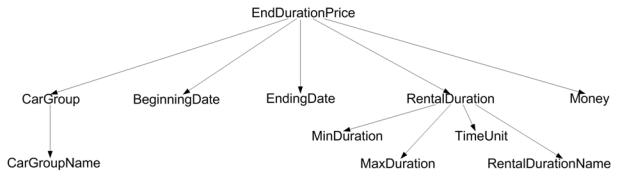

Figure 2: FD-tree for `EndDurationPrice`

Concept	#Dimensions	#Measures
DamageCost	81	3
Prepared	81	3
AssignedCar	80	3
LateReturn	78	5
PaidWithPointsRental	74	4
ClosedRental	74	4
EarlyReturn	74	4

Table 1: Some results of the first task of AMDO

(or properties in OWL notation). We denote concepts by uppercase letters from the beginning of the alphabet (such as A and B) and sets of concepts by uppercase letters from the end of the alphabet (such as Y and Z).

DEFINITION 4.1. *We say that a set of concepts* Z *is a* Super-ID *(SID) of a concept* A*, if there is an injective function from* A *to* Z *(i.e., a mandatory one-to-one relationship).*

Note that we do not ask for a mandatory participation of Z in A (i.e., a bijective function), since some values of the SID could not have a correspondence into the identified concept. This is sound with relational model literature, where a *superkey* is defined as a specific kind of FD. Moreover, according to the relational model assumptions, each relation row is supposed to represent a different instance [6], giving rise as a whole to a one-to-one relationship (furthermore, since a CK does not allow *NULL* values, it is also mandatory). This definition would be equivalent or give rise to others previously introduced in the literature, like the generic *one-to-one relationships* introduced in [5] or the *reference mode* (mandatory one-to-one relationships) in ORM [12], as well as the *base* concept in [1] and *group by set* in [11], specifics for MD modeling.

Def. 4.1 entails that both, the concept identified functionally depends on the SID, and the SID functionally depends on the concept:

PROPOSITION 4.2. *A set of concepts* Z *is a SID of a given concept* A *if and only if* Z *uniquely or functionally determines the values of* A *(i.e.,* Z $\rightarrow$ A*) and* A *functionally determines the values of* Z *(i.e.,* A $\rightarrow$ Z*).*

Note the benefits of this proposition: we will search the SID of A only among the attributes determined by A. When looking for IDs, the searching space is formed by all the attribute combinations up to size N, where N is the number of attributes in the database (i.e., 2^N combinations), but by using ontological knowledge we reduce the searching space to 2^P, where P is the number of concepts functionally dependent on A.

To do this, we require the asserted FDs in the domain ontology. We got them from the first task in AMDO method, but we could use any other generic algorithm. Anyway, for each domain concept we need a *tree of FDs* (FD-tree). Once we have the FD-tree of A, we aim to find the set of concepts Z such that Z is a SID of A. According to Prop. 4.2, we only need to generate combinations of concepts among those functionally determined by A.

Figure 2, shows an example of FD-tree for the `EndDurationPrice` concept of the EU-Car Rental ontology (see [10]), that we will use throughout this paper. It represents the final price charged to the customer's account for the car renting. As shown in the figure, it has 10 FD's: the `car group` (i.e., kind of car rented) and the `car group name`, the `beginning` and `ending` date of the rental agreement, the final price (i.e., `money`) and the rental agreement `duration` (which consists of the `rental duration name` and a `time unit` used to express the `minimum` and `maximum duration` allowed for that rental). All SIDs of `EndDurationPrice` are combinations of these concepts represented in the figure. Note that it is truly a tree, because despite `EndDurationPrice` has two associations to `Date` concept (i.e. `BeginningDate` and `EndingDate`), this generates two nodes in the FD-tree. We will use the term *Fact-children* to denote those concepts in the first level of the FD-tree (in our example: `CarGroup`, `BeginningDate`, `EndingDate`, `Duration` and `Money`).

4.1 Necessary Conditions

A naive approach for discovering IDs would entail generating all the combinations of potential dimensions in our searching space (i.e., 2^P combinations) and sample data to verify them. However, despite we have considerably reduced the searching space, we may still have computational problems for concepts having many FDs, since the searching space is still exponential. For example, in a middle-sized ontology like the EU-Car Rental, after the first task of AMDO (Section 3), we obtained concepts with more than 80 FDs (see the second column of Table 1). Furthermore, querying the data may be expensive for large tables. For this reason, we further exploit the conceptual knowledge we have before verifying ID hypotheses with data. Specifically, we take advantage of the well-known FD theory.

Given a set of FDs F, a *minimal cover* [2] of F is a set F' of FDs such that (being C a single concept):

(i) Each dependency in F' has the form $Z \rightarrow C$,
(ii) $F' \equiv F$,
(iii) no proper subset of F' implies F and
(iv) for each dependency $Z \rightarrow C$ in F' there is no $W \subset Z$ such that $F \models W \rightarrow C$.

In our approach, for every ontology concept A, we define F as the set of FDs of the kind $Z \rightarrow A$, (where Z is compound of concepts in the FD-tree of A). Essentially, we look for a minimal cover of F, because we aim to minimize the number of queries posed to the database (i.e., it is the minimum set of FDs to be verified as SIDs with data). The rest of FDs in F could, if necessary, be generated and verified from F' in polynomial time by the *Armstrong axioms* [19] (i.e., an FD of the kind $Z \rightarrow A$ holds if and only if FD $\in F'^{+}$). Nevertheless, as discussed later, we will not be interested in this kind of FDs either, since they are not minimal and thus, they are not IDs (in the relational model, only *minimal* sets of attributes that uniquely identify the whole *tuple* use to be of interest).

Prop. 4.2 guarantees (i). As discussed in previous section, the FDs we may find in an ontology are of the kind $A \rightarrow B$, (where A and B are concepts), and in our algorithm

we only generate combinations of concepts in the left-hand-side of the FD (i.e., LHS multi-attribute FDs). Regarding (ii), F will be equivalent to the set of FDs determining A that we can infer from knowledge contained in the ontology (from where we compute the initial knowledge that guides the search) and data (from where verify combinations proposed). Thus, if an ID cannot be inferred from the ontology and verified with data, we will not be able to identify it ([20] contains the proofs that our algorithm is complete with regard to knowledge captured in the ontology and data). Finally, (iii) and (iv) guarantee that the set of FDs in F' is minimal. Note that these two conditions are desirable for our purpose, as they avoid redundancy of IDs. Consequently, those FDs in the minimal cover are the only ID candidates to be considered, among all possible SIDs.

Since testing over data is really expensive, we introduce the novelty of using the ontological knowledge to only generate and test those concepts and combinations that fulfill the three necessary conditions introduced below (derived from (iii) and (iv) and *Armstrong axioms*, see [2]). These properties apply to IDs and FDs (note that FDs come from the ontological knowledge, not needing to access instances).

PROPOSITION 4.3. *Let* Z *and* W *be sets of concepts functionally dependent on a given concept* A *(i.e.* A→Z *and* A→W*). If* $Z \subseteq W$ *and* Z *is an ID of* A *then,* W*, despite functionally determining* A*, is not considered an ID, since it is not minimal; there exists a subset of* W *(i.e.,* Z*) that is already an ID of* A.

This proposition is directly formulated from (iv). Intuitively, this enforces the minimality property of IDs. If one concept (or combination) is found to be an ID, we do not need to check all other combinations containing it. For example, it is useless to check whether {`RentalDuration, BeginningDate, Money`} is an ID of `EndDurationPrice` or not, if {`RentalDuration, BeginningDate`} is known to be an ID.

PROPOSITION 4.4. *Let* Z *and* W *be two sets of concepts functionally dependent on a given concept* A *(i.e.* A→Z *and* A→W*). If two concepts* $B \in Z$ *and* $C \in W$ *exist such that* $B \rightarrow C$ *or* $C \rightarrow B$*, then* ZW *is not an ID of* A.

Intuitively, this property says that the combination ZW must not be checked if those concepts in Z and W are not pairwise independent. For example, pair {`RentalDuration, MinDuration`} must not be considered as a potential ID, since `RentalDuration` → `MinDuration`. Checking {`RentalDuration`} and {`MinDuration`} separately would be enough.

PROPOSITION 4.5. *Let* W *be a set of concepts functionally dependent on a concept* A *(i.e.* A→W*), and* C *a concept such that* $C \in W$*. Let* $\mathcal{I}$ *be the set of intermediate concepts giving rise to the to-one path from* A *to* C *(*$\forall C_i \in \mathcal{I} : A \rightarrow C_i \wedge C_i \rightarrow C$*). If* W *is an ID of* A*, then for each* $C_i \in \mathcal{I}$*,* $(W - \{C\}) \cup \{C_i\}$ *is an SID of* A*. We call* $(W - \{C\}) \cup \{C_i\}$ *an intermediate set of* W.

Intuitively, if one concept (or combination) is not an ID, those functionally dependent on it will not be an ID, either. Since `RentalDuration` → `MinDuration`, if {`EndingDate, RentalDuration`} is known not to be an ID then, it is not possible for {`EndingDate, MinDuration`} to be an ID.

Only those combinations satisfying all the conditions are *feasible IDs* and have to be verified with data (proofs in [20]).

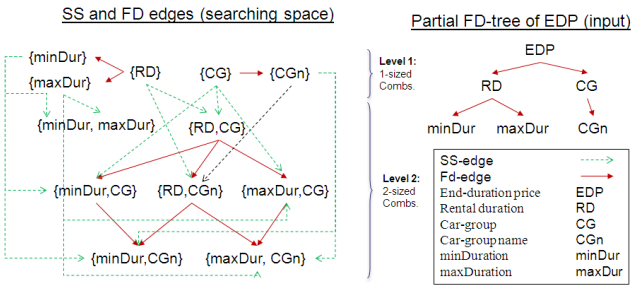

Figure 3: Searching space for `EndDurationPrice` given a partial FD-tree

Thus, we only generate and verify with data those combinations that fulfill them. In this way, we reduce drastically the searching space and the number of combinations to be checked against the database (i.e., minimizing the number of queries posed to the DBMS).

4.2 Searching Space

As discussed, a naive approach for discovering IDs would be to generate all the combinations of concepts and for each one of them query the data to test whether they have repetitions or not. Clearly, this is unfeasible due to the exponential number of potential IDs each fact may have (second column in Figure 1 shows the exponents in our case study). For this reason, we further exploit the properties we have at the conceptual level before verifying hypothesis with data.

Our searching space can be characterized as a directed graph like the one shown in the left side of Figure 3. Two combinations of dimensions in this graph can be related by two different kinds of edges:

SS-edges (Subset) link two nodes of size i and $i+1$ such that the first is a subset of the second one. For instance, the edge between {`RD`} and {`RD, CG`} (we say that {`RD, CG`} is an *SS-descendant* of {`RD`}). Note that these edges link combinations in consecutive depth levels. Therefore, a 3-sized combination such as {`minDur, RD,CG`} would be directly related to {`minDur,maxDur`}, {`minDur, CG`}, and {`maxDur, CG`} and only transitively to {`minDur`}, {`maxDur`}, or {`CG`}, since they are not placed in consecutive depth levels.

FD-edges (Functional Dependency) link two nodes of the same size such that there is *exactly one* concept in the first one functionally determining one concept in the second one (they are derived from the ontology as explained in [24]). For instance, {`RD,CG`} is related to {`RD,CGn`}, since `CG` → `CGn` (we say that {`RD, CGn`} is an *FD-descendant* of {`RD,CG`}). Note that these edges directly relate combinations of the same size where only one concept changes. For instance, {`RD, CG`} is not directly related to {`maxDur, CGn`} despite `CG` → `CGn` and `RD` → `maxDur`, since we must substitute two concepts of {`RD,CG`} to obtain {`maxDur,CGn`}. Indeed, they are related by transitivity.

In our example, we show a piece of the searching space for `EndDurationPrice` (EDP). It is based on the FD-tree of its dimensions. Note that for the sake of simplicity, we use a partial FD-tree of only five concepts (right side of Figure 3). Moreover, we only draw the first two depth levels of the

```
function seek_IDs (Concept A, FdTree M) returns Set<Set<Concept>>

1. Set<Concept> Comb;
2. Ordered Set<Set<Concept>> Candidates_Sets, Feasible_IDs;
3. Set<Set<Concept>> IDs := {};
4. int i:=1; Feasible_IDs := Get_Children(A,M);
5. while(Feasible_IDs != ∅)
   (a) Candidates_Sets := {};
   (b) Comb := Get_First_Combination(Feasible_IDs);
   (c) while(Comb != null)
       i. if(determines(Comb,A)) then
          A. IDs ∪= Comb;
          B. if (Has_FD-Descendants(Comb,M)) then
                Feasible_IDs ∪= Gen_Comb_by_FD(Comb, IDs, M);
       ii. else Candidates_Sets ∪= Comb;
       iii. Feasible_IDs -= Comb;
       iv. Comb := Get_Next_Combination(Feasible_IDs);
   (d) i++;
   (e) Feasible_IDs := Gen_Comb_by_SS(i, IDs, Candidates_Sets, M);
6. return IDs;
```

Figure 4: Algorithm finding Interesting Dimensions

searching space (left side of Figure 3). Talking about depth levels is meaningless in most graphs, but in this case we can still talk about graph depth levels according to SS-edges. On level i we depict those combinations of size i, and SS-edges link combination in two different levels. Inside a depth level we only find FD-edges, and vice-versa.

5. AN ALGORITHM TO DISCOVER IDS

SS and FD-edges introduce a *partial order* in the searching space that our algorithm will follow when generating combinations. It takes advantage of previous generated and tested combinations to decide if we have to explore further alternatives according to the necessary conditions introduced in Section 4.1 (notice that SS-edges will be either explored or pruned based on Props. 4.3 and 4.4 whereas FD-edges will be either explored or pruned based on Prop. 4.5). Three sets are used in the iterations:

Feasible_IDs: In the i^{th} iteration, this set represents those combinations of size i, satisfying the three properties introduced in Section 4.1.

Candidate_sets: In the i^{th} iteration, this set contains those *feasible IDs* of size i refuted as IDs by querying data.

IDs: In the i^{th} iteration, this set contains those *feasible IDs* of size up to i verified as IDs by querying data.

The algorithm (sketched in Figure 4) has two inputs: a fact A we are looking IDs for, and its FD-tree. Given them, it starts considering each Fact-child of A as a *Feasible ID* (Step 4). Thus, these are verified (see Step 5(c)i and Section 5.1) to see if, according to data they identify A. This is sound, since Fact-children generate unary sets trivially fulfilling the necessary conditions, because they do not have proper subsets different form the empty set (and thus, we can directly consider them *feasible IDs*). If any of them is an ID, it is added to the *IDs* set (Step 5(c)iA). Otherwise, if it is not, it is added to the *Candidates_sets* (Step 5(c)ii).

Note that we only explore the FD-descendants of a combination Z if this is an ID for A (see Step 5(c)i and Section 5.1). This is a direct application of Prop. 4.5: the FD-descendants of a given combination Z are not an ID for A if Z is not and ID for A. A combination will be generated by *gen_comb_by_FD* function iff all its direct FD-ancestors are IDs for A (i.e., {`minDur,CGn`} will only be generated if {`minDur,CG`} and {`RD,CG`} are in *IDs* set). Note that we only need to explicitly check the FD-parents, as other ancestors already fulfill Prop. 4.5 transitively (all were generated either in the *gen_comb_by_FD* function or in *gen_comb_by_SS*, and both functions guarantee that new combinations generated do fulfill the three necessary conditions).

The number of new combinations added to the *feasible_IDs* by applying *gen_comb_by_FD* for each ID found, is, at most, the number of direct FD-descendants the ID has. For instance, if {`RD`,`CG`} is an ID, and restricted to the partial FD-tree in Figure 3, then we would add to the *feasible_IDs*: {`minDur`,`CG`}, {`RD`,`CGn`}, and {`maxDur`,`CG`}. Each one of the new combinations generated satisfy Props. 4.3, 4.4, and 4.5 and for this reason we directly add them to the *feasible_IDs* (thus, each new combination is eventually verified as an ID). If any of them results to be an ID then, we apply again *gen_comb_by_FD* and explore its FD-descendants.

Note that a combination generated by *gen_comb_by_FD* that is refuted as an ID (i.e., *determines* function returns false and therefore, it is queued in the *Candidate_sets*) may give rise to IDs of size bigger than i that involve other orthogonal concepts (follwing SS-descendants in Step 5e). For example, if {`minDur`,`CGn`}, and {`maxDur`,`CGn`} are in the candidate sets, they will produce {`minDur`,`maxDur`,`CGn`}. In general, function *gen_comb_by_SS* (see Figure 5) generates $(i+1)$-sized combinations chosen among the SS-descendants of the i-sized candidate sets obtained in the previous iteration. A feasible ID of size *i+1* is only generated if it fulfills those properties in Section 4.1 (as explained in Section 5.2). The algorithm iterates until we are not able to generate feasible IDs of size *i+1*.

5.1 The *determines* Function

This function is called when the three necessary conditions are guaranteed (i.e., we have identified a *feasible ID*). Then, we verify if this combination determines fact A by querying data, which is exemplified with relational queries, but may use any other kind of repository. Prior to query the instances, we first introduce a final pruning rule:

PROPOSITION 5.1. *Let* Z *be a feasible ID, it is able to identify all instances of* A *if the cardinality of* A *is less than (or equal to) the product of the cardinalities of the concepts in* Z *(i.e.,* $\prod_{Z_i \in Z} |Z_i| \geq |A|$*)*

Note that this pruning rule discards combinations by just querying the RDBMS catalog, as follows (Oracle syntax):

```
SELECT NUM_ROWS FROM USER_TABLES WHERE TABLE_NAME = t;
SELECT NUM_DISTINCT FROM USER_TABS_COLS WHERE TABLE_NAME = t AND
COLUMN_NAME = c;
```

Being t the name of a table and c that of a column, if the ontology concept maps to a relational table then by means of the first query we get the cardinality of t, and if the ontology concept maps to a relational attribute by means of the second query we get the number of different values it has. Those combinations satisfying this rule are still candidates to be an ID, and we verify it by the following query (Oracle syntax):

```
SELECT "ID" FROM DUAL WHERE NOT EXISTS(
   SELECT attrSet FROM tables WHERE joinConds
   GROUP BY attrSet HAVING COUNT(*) > 1)
```

Where `DUAL` is the `dummy` table in Oracle, `attrSet` are the attributes forming the *feasible ID* to be verified, `tables` the list of tables containing those attributes and `joinConds` the join clauses needed to join tables involved in the query. If we are able to find two rows with the same values for the ID hypothesis then, according to data, it is not an ID.

```
function Gen_Comb_by_SS (int i, Set<Set<Concept>> IDs, Ordered
Set<Set<Concept>> Candidates_Sets, FdTree M) returns
Set<Set<Concept>>

  1. Set<Set<Concept>> Combs := {};
  2. For(int j = 0; j < sizeof(Candidates_Sets); j++)
       (a) CS1 := get_element(Candidates_Sets, j);
       (b) For(int z = j+1; z < sizeof(Candidates_Sets); z++)
             i. CS2 := get_element(Candidates_Sets, z);
            ii. if (|CS1∩CS2|=(i-1) and (i != 2 or independent(CS1,CS2))
                and (check_Subsets(CS1,CS2, IDs, Candidates_Sets))) then
                  A. Combs ∪= {CS1∪CS2};
  3. return Combs;

function check_Subsets (Set<Concept> CS1, Set<Concept> CS2,
Set<Set<Concept>> IDs, Set<Set<Concept>> Candidates_Sets) returns
Boolean

  4. For each(subSet in Generate_All_i_Subsets(CS1, CS2)) do
       (a) if(subSet in IDs) then return false;
       (b) else if(subSet not in Candidates_Sets) then
             i. if(all_Fact-children(subSet)) then return false;
  5. return true;
```

Figure 5: Generating (i+1)-sized combinations

5.2 The *gen_comb_by_SS* Function

Once the *i*-sized combinations have been verified (i.e., either proved to be IDs, and thus added to the *IDs set*, or refuted, and thus, added to the *candidate sets*), the function *gen_comb_by_SS* (see Figure 5) generates *(i+1)*-sized combinations from the *i*-sized *candidate sets* obtained in the previous iteration. It looks for pairs of candidate sets having *i*-1 concepts in common (Step 2(b)ii). From every pair identified sharing *i-1* concepts, a *(i+1)*-sized combination is generated (Step 2(b)iiA) fulfilling the following properties:

Prop. 4.3: guaranteed in Step 2(b)ii by the *check_Subsets* function. This function generates all the *i*-sized subsets of the current *(i+1)*-sized combination treated, and verify them as follows:

(a) If an *i*-sized subset of the *(i+1)*-sized set is in the *IDs* set (Step 4a) then, it must not be considered a *(i+1)*-sized *feasible ID*, since it is not minimal.

(b) If an *i*-sized subset is in the *Candidate_sets* (Step 4b), the *(i+1)*-sized combination can still be minimal and some of it concepts have been checked to be pairwise orthogonal (eventually, all will be checked in the loop).

(c) Alternatively, due to our pruning rules, it may happen that a subset is neither in the *Candidate_sets* nor in the *IDs* set. If the subset is only compound of Fact-children (Step 4(b)i), the *(i+1)*-sized combination must be refuted, since our algorithm checks all Fact-children combinations fulfilling Props. 4.3 and 4.4.

(d) Otherwise, the i-sized subset is a transitive FD-descendant (remember that *gen_comb_by_FD* function only generated direct FD-descendants) of an *i*-sized *feasible ID* refuted with data and therefore, fulfilling Props. 4.3 and 4.4.

Prop. 4.4: guaranteed, for 2-sized combinations, in Step 2(b)ii (i.e., checking that they are not present in the FD-tree of one another) and, for bigger combinations, similarly to 4.3 in *check_Subsets* function.

Prop. 4.5: actually guaranteed by *gen_comb_by_FD* function and *(i+1)*-sized combinations coming from the union of two *i*-sized feasible IDs.

For example, suppose that we try to combine {`BeginningDate, MinDuration`} and {`BeginningDate, Money`} to produce {`BeginningDate, MinDuration, Money`}. Then, function *check_Subsets* will generate its three 2-sized subsets: {`BeginningDate, MinDuration`}, {`BeginningDate, Money`} and {`MinDuration, Money`}. According to (a), if any of them is an ID then, the 3-sized combination is not generated (since it is not minimal). Oppositely, according to (b), if all of them are in the *candidate sets*, we can guarantee that the 3-sized combination is minimal and it is generated. Consider now that {`BeginningDate, Money`} and {`BeginningDate, MinDuration`} are neither in the *candidate sets* nor in the *IDs set*. According to (c), since the first one is compound of Fact-children, if fulfilling Props. 4.3 and 4.4, it should have been in either *IDs* or *candidate sets* (since our algorithm is exhaustive for Fact-children) and thus, it means that either {`BeginningDate`} or {`Money`} is an ID. In the second case, since {`BeginningDate, MinDuration`} is an FD-descendant of {`BeginningDate, RentalDuration`} it does not invalidate the 3-sized combination. The reason is that {`BeginningDate, RentalDuration`} is a *feasible ID* refuted with data. Consequently, since it was refuted as an ID, according to Prop. 4.5, we could foresee that {`BeginningDate, MinDuration`} would not be an ID and then, it was not even generated by the *gen_comb_by_FD* function (had it been generated by the *gen_comb_by_SS* function, this subset would have been included in the *IDs set* or *candidate sets* and thus, considered by either (a) or (b)). However, this set fulfills Props. 4.3 and 4.4 and therefore, it does not invalidate the 3-sized combination.

6. CASE STUDY

In this section, we introduce results got after carrying out our algorithm over the EU-Car Rental case study [10]. This ontology refers to a car renting domain [1] and it has 65 concepts and 170 relationships. For each concept, we computed its asserted FDs (from knowledge asserted in the ontology) and later, by means of the algorithm introduced in Section 5, its *feasible IDs*. As a result, each concept has an average of 31.83 concepts in the FD-tree (i.e., in our algorithm we have an average searching space of $2^{31.83}$ combinations). Among concepts in the FD-tree, an average of 6.67 are Fact-children (i.e., the average value of combinations we start with in the first iteration of our algorithm). When computing IDs we get an average of 156.53 *feasible IDs* per concept (i.e., we will query the database 156.53 times in average; in front of the 3,817,550,246 times if we would have generated all the combinations in the searching space, i.e., $2^{31.83}$). In general, considering all the queries posed to the database for all the concepts we have a total of 10,018 queries, of which 15% are answered by just querying the catalog (see Section 5.1) and the rest, by queries over data. This is the minimum number of queries we must pose to find all the IDs of the domain (exactly the combinations fulfilling the conditions stated in Section 4). Furthermore, it is interesting to realize that about 30% of *n*-sized *feasible IDs* are generated by combining *(n-1)*-sized *candidate sets* whereas the rest are generated by following FD-edges. The execution time of our algorithm is insignificant in front of the cost of querying data. Indeed, all the *feasible IDs* for all the concepts were generated in less than 10 seconds (Intel Core2Duo 1.33 GHz processor, 1.99 GB of RAM running a JVM 1.4.2 accessing the ontology through Protégé-OWL API [9]).

[1] The whole ontology is available in OWL DL notation at www.lsi.upc.edu/∼oromero/EUCarRental.owl

7. CONCLUSIONS

We have proposed an algorithm for finding fact IDs (i.e., composite keys) using domain ontologies. We take advantage of knowledge captured in the ontology to generate IDs hypotheses that are later verified with data. Unlike previous approaches that addressed this task at the data level, we benefit from ontological knowledge that allows to better depict and prune the searching space. As consequence, our approach does not completely rely on data and it opens new perspectives for data quality processes. We have presented the feasibility of our method by means of the statistics raised by the implementation of our algorithm over a case study (soundness and completeness proofs with regard to knowledge captured in the domain ontology and the data are found in [20]).

8. REFERENCES

[1] A. Abelló, J. Samos, and F. Saltor. **YAM**^{2} (Yet Another Multidimensional Model): An Extension of UML. *Information Systems*, 31(6):541–567, 2006.

[2] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, 1995.

[3] A. Bonifati, F. Cattaneo, S. Ceri, A. Fuggetta, and S. Paraboschi. Designing Data Marts for Data Warehouses. *ACM Trans. Softw. Eng. Methodol.*, 10(4):452–483, 2001.

[4] D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, and R. Rosati. Path-Based Identification Constraints in Description Logics. In *11th Int. Conf. on Principles of Knowledge Representation and Reasoning*, pages 231–241. AAAI Press, 2008.

[5] P. P. S. Chen. The Entity-Relationship Model: Toward a Unified View of Data. *ACM Transactions on Database Systems*, 1(1):9–36, 1976.

[6] E. F. Codd. *The Relational Model for Database Management, Version 2*. Addison-Wesley, 1990.

[7] J. Demetrovics and V. D. Thi. Some Remarks On Generating Armstrong & Inferring Functional Dependencies Relation. *Acta Cybernetica*, 12(2):167–180, 1995.

[8] P. A. Flach and I. Savnik. Database Dependency Discovery: A Machine Learning Approach. *AI Commun.*, 12(3):139–160, 1999.

[9] S. C. for Biomedical Informatics Research. Protégé-OWL API (last access 17/12/2009). `protege.stanford.edu/plugins/owl/api`.

[10] L. Frías, A. Queralt, and A. Olivé. EU-Rent Car Rentals Specification. Technical report, "Dept. de Llenguatges i Sistemes Informàtics", 2003. `www.lsi.upc.edu/dept/techreps/llistat_detallat.php?id=690`.

[11] M. Golfarelli and S. Rizzi. *Data Warehouse Design*. McGraw-Hill, 2009.

[12] T. Halpin and T. Morgan. *Information Modeling and Relational Databases*. Morgan Kauffman, 2008.

[13] M. R. Jensen, T. Holmgren, and T. B. Pedersen. Discovering Multidimensional Structure in Relational Data. In *6th Int. Conf. on Data Warehousing and Knowledge Discovery*, pages 138–148. Springer, 2004.

[14] W. M. Lim. Discovery of Constraints from Data for Information System Reverse Engineering. In *2nd Australian Software Engineering Conf.*, pages 39–48. IEEE, 1997.

[15] M. Kantola, H. Mannila, K.-J. Räihä, and H. Siirtola. Discovering Functional and Inclusion Dependencies in Relational Databases. *Int. J. of Intelligent Systems*, 7(7):591–607, 1992.

[16] A. Olivé. On the Role of Conceptual Schemas in Information Systems Development. In *9th Int. Conf. on Reliable Software Technologies*. Springer, 2004.

[17] C. Phipps and K. C. Davis. Automating Data Warehouse Conceptual Schema Design and Evaluation. In *4th Int. Workshop on Design and Management of Data Warehouses*, volume 58, pages 23–32. CEUR-WS.org, 2002.

[18] A. Poggi, D. Lembo, D. Calvanese, G. De Giacomo, M. Lenzerini, and R. Rosati. Linking Data to Ontologies. *J. on Data Semantics*, 10:133–173, 2008.

[19] R. Ramakrishnan and J. Gehrke. *Database Management Systems*. McGraw Hill, 2003.

[20] O. Romero. *Automating the Multidimensional Design of Data Warehouses*. PhD thesis, Universitat Politècnica de Catalunya, Barcelona, Spain, 2010. `www.tesisenxarxa.net/TESIS_UPC/AVAILABLE/TDX-0528110-134628//TOR%M1de1.pdf`.

[21] O. Romero and A. Abelló. Automating Multidimensional Design from Ontologies. In *Proc. of ACM 10th Int. Workshop on Data Warehousing and OLAP*, pages 1–8. ACM, 2007.

[22] O. Romero and A. Abelló. A Survey of Multidimensional Modeling Methodologies. *Int. J. of Data Warehousing and Mining*, 5(2):1–23, 2009.

[23] O. Romero and A. Abelló. Automatic Validation of Requirements to Support Multidimensional Design. *Data & Knowledge Engineering*, 69(9):917–942, 2010.

[24] O. Romero, D. Calvanese, A. Abelló, and M. Rodriguez-Muro. Discovering Functional Dependencies for Multidimensional Design. In *ACM 12th Int. Conf. on Data Warehousing and OLAP*, pages 1–8. ACM, 2009.

[25] Y. Sismanis, P. Brown, P. J. Haas, and B. Reinwald. GORDIAN: Efficient and Scalable Discovery of Composite Keys. In *32nd Int. Conf. on Very Large Data Bases*, pages 691–702. ACM, 2006.

[26] I. Song, R. Khare, and B. Dai. SAMSTAR: A Semi-Automated Lexical Method for Generating STAR Schemas from an ER Diagram. In *10th Int. Workshop on Data Warehousing and OLAP*, pages 9–16. ACM, 2007.

[27] C. Soutou. Relational Database Reverse Engineering: Algorithms to Extract Cardinality Constraints. *Data & Knowledge Engineering*, 28(2):161–207, 1998.

[28] R. Wieringa and W. de Jonge. Object Identifiers, Keys, and Surrogates: Object Identifiers Revisited. *Theory & Practice of Object Systems*, 1(2):101–114, 1995.

[29] R. Winter and B. Strauch. A Method for Demand-Driven Information Requirements Analysis in DW Projects. In *36th Annual Hawaii Int. Conf. on System Sciences*, pages 231–239. IEEE, 2003.

[30] D. Yeh, Y. Li, and W. C. Chu. Extracting E-R diagram from a table-based legacy database. *J. of Systems and Software*, 81(5):764–771, 2008.