Under consideration for publication in Knowledge and Information Systems

Ontology Driven Search of Compound IDs

Alberto Abelló and Oscar Romero

Dept. Enginyeria de Serveis i Sistemes d'Informació
Univ. Politècnica de Catalunya
E-08034 Barcelona, Spain
{aabello|oromero}@essi.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* identifiers (i.e., guaranteeing the entity identity according to the semantics of the universe of discourse), 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 purely at the data level may not be affordable. In this paper, we propose an approach for discovering meaningful identifiers driven by *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 identifier 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.

Keywords: Identifiers, Compound keys, Reengineering, Ontologies

1. Introduction

Object identification is a crucial step in most information system engineering processes. 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 (Wieringa and de Jonge, 1995). As example, the uniqueness of the system controlled surrogates (a common way to implement identifiers, IDs from now on, in relational databases) is useless

Received Nov 24, 2010
Revised Mar 08, 2011
Accepted May 08, 2011

RentalAgreement	
id: integer	basicPrice: Money
rentalDurationTimeUni: Period	bestPrice: Money
lastModification: Date	rentalDurationName: String
minimumRentalDuration: Integer	maximumRentalDuration: Integer
pickUpExpectedTime: Date	assignedCar: String
pickUpBranchName: String	pickUpBranchType: String
pickupBranchCountry: String	pickUpBranchMechReq: String
pickUpBranchEmissionEq: String	pickUpBranchCarTax: Float
pickUpBranchServiceDepotName: String	pickUpBranchServiceDepotCapacity: String
dropOffBranchName: String	dropOffBranchType: String
dropOffBranchCountry: String	dropOffBranchMechReq: String
dropOffBranchEmissionEq: String	dropOffBranchCarTax: String
dropOffBranchServiceDepotName: String	dropOffBranchServiceDepotCapacity: String
customerName: String	customerId: String
customerBirthDate: Date	customerAddress: String
customerTelephone: String	customerDrivingLicenseNumber: Integer
...	

Figure 1. Example of identifiers

for data quality, because they are internal IDs with no semantics regarding the universe of discourse. Thus, surrogates remain useless in any context where instances extracted from different systems need to be asserted as equal. For example, consider the table depicted in figure 1. The relation identifier is the `id` attribute (a surrogate). In this case, the `{pickupExpectedTime,customerId}` or even `{pickupExpectedTime,assignedCar}` would be a much better identifier, since every person is supposed to have only one rental agreement for a given date and, from the point of view of the company, they can only assign the same car to one person the same day. Meaningful IDs (compound by the attributes -i.e., features- of the entities) guarantee the entity identity according to the semantics of the universe of discourse (or *system domain*), and they 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. In this paper, we will focus on the latter to outline the relevance of providing meaningful identifiers on such systems and exemplify the multipurpose characteristic of our approach by adapting it to two different usages in this kind of environment.

According to (Golfarelli and Rizzi, 2009), a DW is a *collection of methods, techniques, and tools used to support knowledge workers -e.g., senior managers, directors, etc.- to conduct data analysis that help with performing decision making processes and improving information resources*. Typically, relevant data for decision making is extracted from the organization data sources, transformed (i.e., cleaned and homogenized) and finally integrated within a huge repository of data (the DW), in what is known as the ETL (Extraction / Transform / Loading) process. The DW provides a single and detailed view of the organization, and it is intended to be exploited by means of the *exploitation tools*, which provide different mechanisms to navigate and perform analysis tasks over the DW. Among the different kinds of exploitation tools, *On-line Analytical Processing* (OLAP) tools have gained relevance in the last years so much so that the concepts of DW and OLAP are now tightly related.

OLAP tools are intended to facilitate information analysis and navigation through the business data based on the dimensional paradigm. The dimensional view of data is distinguished by the *fact / dimension* dichotomy, and it is characterized by representing data (i.e., the fact of interest) as if placed in an *n-dimensional space* (with as many axis as dimensions of analysis of interest). This paradigm allows to easily understand and analyze data showing the different points of view from where a subject can be analyzed. In consequence, the

dimensional model fits for non-expert users like knowledge workers. For example, a typical fact of interest would be the business `sales`, whereas its typical dimensions of analysis would be the `item` sold, `where` it was sold (i.e., the place) and `when` (i.e., the time). One fact and several dimensions to analyze it give rise to what is known as the *data cube*[1] and the conceptual representation of this data structure is known as *dimensional schema*.

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 dimensional 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 (to some extent) 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 for the organization. Anyway, it will always be up to the users to select those facts of interest to their business. For example, note that what is a fact for a user may be just an analysis dimension to another.

In this scenario, it is important to provide tools like the one presented in this paper, because DW systems design will benefit from knowledge about IDs in the following ways: (i) identifying facts (since good candidates are precisely those with compound IDs; i.e., those thought as a set of entity attributes), (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 dimensional spaces where data can be placed (in case of different compound IDs for the same fact), (iv) avoid sparse cubes (since, by definition, IDs are minimal sets), and (v) find overlapping cubes (to potentially improve the physical design by storing together those data that can be easily analyzed together). Moreover, meaningful IDs play a crucial role to extract the instances from several, potential heterogeneous sources, transform and homogenize them and eventually load them in the DW. All in all, discovering meaningful IDs is essential to support the DW design task.

In the context of DW, current semi-automatic approaches mostly carry out the design task from relational OLTP (*On-Line Transaction Processing*) systems, assuming that a RDBMS is the most common kind of data sources we may find, and taking as starting point a relational schema (i.e., starting from a logical design). Thus, they aim at deriving dimensional schemas from the relational sources. As previously discussed, in dimensional design the dimensions produce the dimensional space (i.e., the analysis perspectives) where the fact will be placed. Conceptually, it entails that the fact is related by means of to-one relationships to the dimensions, which in turn *identify* the fact. Discovering this kind of relationships is crucial in the design of the DW, and the most common way to represent them at the relational level is by means of "foreign" (FK) and "candidate key" (CK) constraints. Briefly, these methods try to produce, from

[1] The data cube refers to the placement of factual data in the dimensional space. And thus, it can be thought as a mathematical function.

the relational sources, *star-shaped schemas* (Kimball, Reeves, Thornthwaite and Ross, 1998), which are the relational counterpart of dimensional schemas where the *central* fact table has not an atomic ID but one compound by *foreign keys* (FKs) pointing to the dimension tables. Thus, discovering such compound IDs is a must.

These dimensional design methods require that the relational schema explicitly captures the to-one relationships (i.e. functional dependencies) existing in the domain. We may find several approaches to discover *functional dependencies* (FDs or "$\rightarrow$") and/or IDs[2] (in the relational context, *candidate keys* (CKs)), see Section 2. Nevertheless, these approaches still 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. For this reason, 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). Consequently, most automated approaches for identifying IDs address this task at the instance level. These methods, however, have various drawbacks: they tend to overlook compound IDs (essential when dealing with dimensional databases, for example), propose solutions that are computationally expensive, and register drops in performance when a large number of attributes or instances are processed.

Due to the prohibitive cost of discovering compound IDs at the instance level, current dimensional design approaches rely on heuristics to identify entities as potential facts and from them, identify dimensional data by identifying FK - CK relationships. In any case, the degree of normalization (ranging from a completely denormalized relation to 3NF) of the relational schema may affect the output quality and performance of current dimensional design methods, since they would need to look for FDs at the instance level, if the logical schema lacks this semantics.

In this paper[3], in order to conduct a more informed search and reduce the output obtained in the analysis of the data sources when discovering IDs, we work over a conceptual formalization of the domain (instead of a logical one). Specifically, we propose an approach to discover meaningful IDs driven by domain ontologies. The software engineering area has claimed to use conceptual representations of the domain on top of systems to have an up-to-date and accurate formalization of the system domain (Olivé, 2004). This approach has given rise to concepts such as OBDA (Ontology-Based Data Access) (Poggi, Lembo, Calvanese, De Giacomo, Lenzerini and Rosati, 2008) and nowadays, ontologies are used in many fields such as data integration, conceptual modeling, the semantic web, among others (Francisco, Gervás and Peinado, 2010; Assawamekin, Sunetnanta and Pluempitiwiriyawej, 2010).

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

[2] Note that the traditional ID concept is just a specific case of FD. For example, see (Abiteboul, Hull and Vianu, 1995).

[3] Short version published in (Abelló and Romero, 2010).

(W3C, 2009), which is a W3C recommendation, do not provide identification mechanisms, and only very expressive DL (that are not suitable for real world applications due to the computational complexity of their reasoning algorithms) incorporate them (Calvanese, De Giacomo, Lembo, Lenzerini and Rosati, 2008). 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 in most (if not all) domains. Furthermore, the fact that most DLs do not consider n-ary relationships makes impossible to assert compound 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 process we follow. Section 4 establishes the foundations, and Section 5 explains the details of the algorithm. Section 6 contains the proofs of its soundness and completeness, while Section 7 exemplifies other cases where it can be used. Finally, Section 8 gives some statistics of a simulation over a middle-sized ontology to show the feasibility of our method. Section 9 concludes the paper.

2. Related Work

Manual discovery of IDs is an unfeasible task for most systems, and current automatic methods need to address this task at the logical or instance level. To avoid missing some important data dependencies, those approaches working at the logical level assume the completeness of the data structures (i.e., all the constraints of the domain of interest are captured at the logical level) or try to infer them by exploiting the semantics of additional structures such as triggers, procedures or transactions (e.g., (Hainaut, Chandelon, Tonneau and Joris, 1993; T. and Zhao, 2004)). The former assumption may not hold and the latter uses to be too complex.

Addressing this issue at the data level (e.g., (Demetrovics and Thi, 1995; Huhtala, Kärkkäinen, Porkka and Toivonen, 1999; King and Legendre, 2003; Lopes, Petit and Lakhal, 2000; M. Kantola, H. Mannila, K.-J. Räihä, and H. Siirtola, 1992; Novelli and Cicchetti, 2001) among others) though, would imply such an effort that may not be worth (as explained in (Jensen, Holmgren and Pedersen, 2004)). 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) it would rise an unaffordable computational complexity (e.g., (Soutou, 1998) generates an exponential number of queries). Indeed, the best of these approaches yields a quadratic computational cost regarding the size of data. Yet they have been proved not to scale properly for big loads of data, which is the most common scenario in DW systems, as pointed out, for example, in (C. Monash, 2008; Golfarelli and Rizzi, 2009). For this reason, this is still a research topic giving rise to new proposals, like (Sismanis, Brown, Haas and Reinwald, 2006; Yao and Hamilton, 2008; Yeh, Li and Chu, 2008). Furthermore,

note that 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).

Specifically, the automation of the DW design process from relational schemas has been thoroughly addressed in the literature (e.g., (Golfarelli, Maio and Rizzi, 1998; Giorgini, Rizzi and Garzetti, 2005; Phipps and Davis, 2002; Jensen et al., 2004; Romero and Abelló, 2010) among others). All these methods rely on a thorough analysis of the relational sources. In these cases, a first step aimed at discovering facts is mandatory. Instead of using the IDs of the tables (which may be missing), these methods propose the usage of heuristics in this task (such as the relation cardinality or number of numerical attributes each table has). From facts identified in this first step, these methods discover dimensional data by means of design patterns based on FK and CK constraints (which again may be missing). In general, these methods avoid sampling data (neither for discovering compound IDs for facts nor to identify dimensional data) and the accuracy of results obtained purely 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, there are only two exceptions to this general rule. On the one hand, (Romero and Abelló, 2010) requires the analytical requirements, expressed in the form of SQL queries, besides the input relational schema. In this way, the join information in the user requirements is used to infer inclusion dependencies and avoid analyzing the data (in other words, requirements overcome the lack of semantics in the relational schema). On the other hand, (Jensen et al., 2004) presents a method aimed at exploiting the instances semantics to complete the relational schema metadata, and give support to the dimensional design. Assuming that the DB does not contain compound keys, this method derives valuable metadata such as functional and inclusion dependencies and key information not stated in the schema. To do so, they take advantage of data mining techniques over the data sources. In short, they assume that a functional or inclusion dependency that holds in the data holds in the schema as well (i.e., two attributes of two different tables are analyzed to know if a to-one relationship holds between them when a FK-CK relationship is not explicitly stated in the schema). Thus, they are able to identify hidden inclusion dependencies, but clearly the cost is computationally high, since they check all pairs of attributes in the DB.

The approach presented in this paper is the first work addressing the issue of discovering meaningful IDs at the conceptual level. Unlike previous approaches, we use ontological knowledge to guide the search and we do not exclusively rely on data. It has two main benefits. On the 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 discovered at the conceptual level refuted by data).

3. Overview

To automate the process as much as possible, we choose ontologies instead of other conceptual formalizations, since ontology languages provide reasoning services that will facilitate the automation of our task. Specifically, we used OWL DL in our implementation. Nowadays ontology languages are widely used in different areas like data integration (Lenzerini, 2002) and the Semantic Web (Berners-Lee, Hendler and Lassila, 2001), but in other areas, like software engineering, UML (OMG, 2010) and ER (Chen, 1976) are the most common choices. In these cases, our approach requires a *pre-process* to generate an OWL DL ontology from the UML or ER diagram. As discussed in the literature (see, for example, (Artale, Calvanese, Kontchakov, Ryzhikov and Zakharyaschev, 2007; Berardi, Calvanese and Giacomo, 2005; Gaševic, Djuric and Devedžic, 2007)), this process can be automated.

Nevertheless, were a domain ontology available, it should be enriched with the end-user requirements in order to properly capture both the domain vocabulary and the data sources, and produce a *user-centered* domain ontology (i.e., an ontology capturing the necessary knowledge about our domain and the underlying sources). In these cases, or even if there is not any conceptual formalization of the domain available we can use integration and reverse engineering techniques for building the needed ontology. For example, (Skoutas and Simitsis, 2007) shows that structured as well as unstructured data stores can be elegantly represented as graphs, and it describes how an ontology for such data stores can be semi-automatically constructed by integrating a domain vocabulary with the data sources' vocabulary. An alternative to the previous approach is presented in (Dánger and Berlanga, 2009). There, a tool for the extraction of complex instances from free text on the Web is presented. The approach is based on non-monotonic processing, and by means of entity recognizers and disambiguators, it adequately helps to create and combine instances and their relations. The complementary works in (Berlanga, Jiménez-Ruiz, Nebot and Sanz, 2010; Nebot and Berlanga, 2009) enable the customized use of available knowledge resources such as thesaurus or ontologies to assist in the annotation process. Both methods above discussed focus on providing support for building tailored, ad hoc ontologies (i.e., user-centered) from large repositories (indeed, both approaches focuses on building ontologies to support integration tasks for data warehousing so they can be regarded as a pre-process of our method). Finally, note that both processes are semi-automatic and thus, they demand some manual workload in order to adapt the automatic output they are able to produce to our scenario and requirements.

To start our process, the user should choose some of the concepts from the ontology. The novelty of our approach to avoid incurring in a high computational cost is that all the concepts functionally determined by current studied concept will be analyzed to check whether they form an ID or not (for further details on the formal definition of ID, we address the reader to Section 4). The different phases are depicted in figure 2.

1. The first phase generates sets of concepts than can give rise to an ID of the concept chosen by the user. They are compound only of concepts that, based on the ontological knowledge available, are functionally determined by the input concept. Moreover, we use those properties introduced in Section 4.1, derived from the well-known FD theory, to avoid the generations of useless

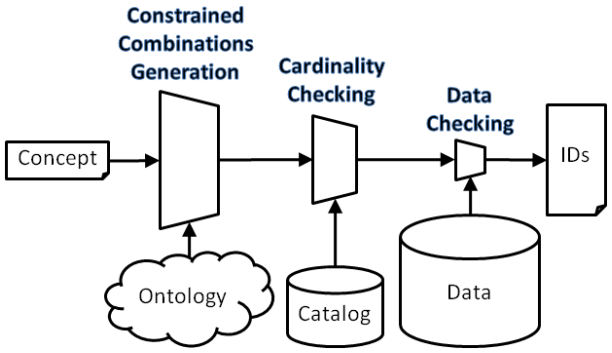

Figure 2. Process Overview

combinations. Thus, just a reduced subset of all possible combinations is generated.
2. Since accessing data can be really expensive, we save it for some combinations by querying only the catalog/metadata of the database/system and comparing the maximum cardinality of the ID against that of the concept.
3. Finally, for those combinations of attributes with enough cardinality to be able to identify the concept, we check whether they actually do it or not accessing the data. We assume that these data are in a relational DBMS, however it could be easily adapted to access any other kind of repository by just providing the sentences equivalent to those SQL statements we show in Section 5.1.
4. The result of the whole process will be a list of possible IDs. This list can be ranked by the maximum cardinality each ID would admit. The closer the cardinality of the ID to that of the concept we want to identify, the more likely it corresponds to a truly meaningful ID.

The IDs identified at this point might be used to support the dimensional design task (as previously discussed in Section 1), evaluate data quality, etc.

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 (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 a complete 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

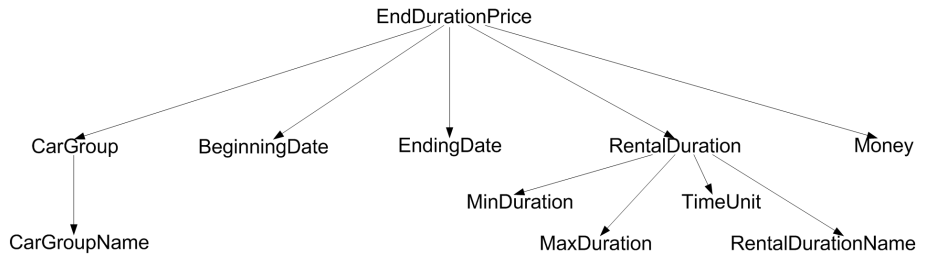

Figure 3. FD-tree for `EndDurationPrice`

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 (Codd, 1990), 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 (Chen, 1976) or the *reference mode* (mandatory one-to-one relationships) in ORM (Halpin and Morgan, 2008), as well as the *base* concept in (Abelló, Samos and Saltor, 2006) and *group by set* in (Golfarelli and Rizzi, 2009), specifics for dimensional modeling.

Def. 4.1 entails that both, the concept identified functionally depends on the SID, and the SID functionally depends on the concept. Thus, we can formulate the following proposition:

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., $A \rightarrow Z$).

Thus, we will search the SID of A only among the attributes determined by A. When looking for SIDs, 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, i.e. for each domain concept we need a *tree of FDs* (FD-tree). Note that retrieving this information from the ontology is cheap in general (linear on the number of assertions, in the worse case) and even cheaper compared to doing it over data. Once we have the FD-tree of A, we aim to find the set of concepts Z in it 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 3, shows an example of FD-tree for the `EndDurationPrice` concept of the EU-Car Rental ontology (see (Frías, Queralt and Olivé, 2003)), 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

Concept	#FD concepts
DamageCost	81
Prepared	81
AssignedCar	80
LateReturn	78
PaidWithPointsRental	74
ClosedRental	74
EarlyReturn	74

Table 1. Number of concepts functionally depending on each concept

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 *root-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, we obtained concepts with more than 80 FDs (see Table 1). Furthermore, querying the data may be expensive for large tables. This becomes more relevant given that the first petabyte DWs have turned into a reality. 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* (Abiteboul et al., 1995) 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* (Ramakrishnan and Gehrke, 2003) (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 (e.g., 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. 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.

In our approach, these two items (i.e., (iii) and (iv)) lead to two necessary conditions a FD must fulfill prior to be verified as an ID with data. Before introducing them, we recall the *Armstrong axioms* used to infer all the FDs that can be computed from a given set F of FDs (i.e., the F closure or F^+) (Abiteboul et al., 1995), since we will need them to proof our propositions:

- (reflexivity) If $Y \subseteq X$, then $X \rightarrow Y$,
- (augmentation) If $X \rightarrow Y$, then $X \cup Z \rightarrow Y \cup Z$
- (transitivity) If $X \rightarrow Y$ and $Y \rightarrow Z$, then $X \rightarrow Z$.

Here, we also introduce the pseudo-transitivity rule (Abiteboul et al., 1995) (that can be easily derived from the three inference rules discussed above), since we will also need it later:

- (pseudo-transitivity) If $X \rightarrow Y$ and $YW \rightarrow Z$, then $XW \rightarrow Z$.

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*. 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. $A \rightarrow Z$ and $A \rightarrow W$). 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. $A \rightarrow Z$ and $A \rightarrow W$). 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 needs not to 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` $\rightarrow$ `MinDuration`. Checking separately {`RentalDuration`} and {`MinDuration`} would be enough.

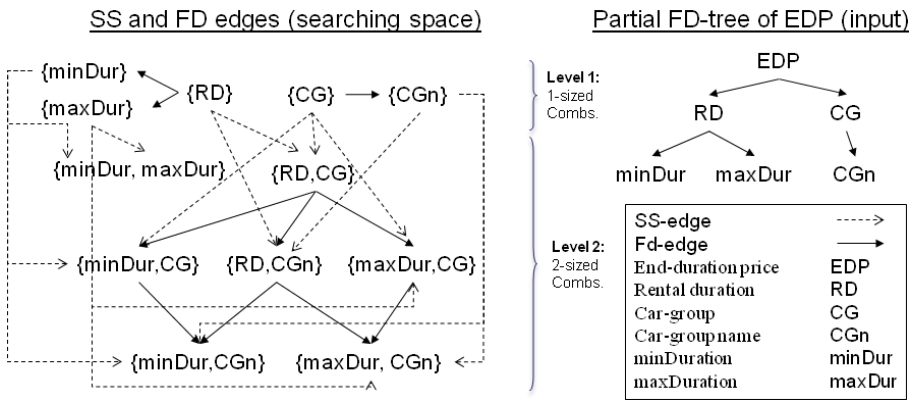

Figure 4. Searching space for `EndDurationPrice` given a partial FD-tree

Proposition 4.5. Let W be a set of concepts functionally dependent on a concept A (i.e. $A \rightarrow W$), 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` $\rightarrow$ `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 Section 6). 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 Table 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 at the left side of Figure 4. 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. 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). Note that for the sake of simplicity, we use a partial FD-tree of only five concepts (right side of Figure 4). Moreover, we only draw the first two depth levels of the searching space (left side of Figure 4)[4]. 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 5) has two inputs: a concept A we are looking IDs for, and its FD-tree. Given them, it starts considering each root-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 root-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.i.A). 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*

[4] 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.

```
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 != Ø)

   (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) Feasible_IDs := Gen_Comb_by_SS(i, IDs, Candidates_Sets, M);
   (e) i++;

6. return IDs;
```

Figure 5. Algorithm finding IDs

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 4, 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 greater than i that involve other orthogonal concepts (following SS-descendants in Step 5.d). 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 6) 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*.

As explained in Section 4.2, the size of the searching space is exponential on the number of concepts in the FD-tree. However, this algorithm is only exhaustive for root-children combinations, and only until it finds an ID. Just when an ID is found the algorithm explores deeper levels in the FD-tree, but at the same time it does not explore more combinations including the attributes being part of this ID. Thus, the worst-case scenario (i.e., the one generating 2^P combinations) is that having P root-children concepts and all of them together resulting in the

only ID (if any). We consider that the larger P is, the more unlikely this is to happen.

For example, executing the algorithm for `EDP`, considering the whole FD-tree in Figure 3 (with 10 concepts), we would generate 53 `feasible IDs` (far from 2^{10}). And after testing only 29 of them with data, we would find among the results {`CG`,`RD`,`EndingDate`} and {`CG`,`RD`,`BeginningDate`}.

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. Below, we exemplify it with relational queries by assuming a RDBMS, but we may consider any other kind of repository or testing method (just replacing the SQL queries by the proper access technology). 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 rule discards combinations by just querying the RDBMS catalog (or system metadata, if available, if other kind of sources are considered) 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;
```

At this point, we need to use the mappings from the ontology to the data sources, which we assume are available. As discussed in section 3, it is important to remark that, if we started working from a generic domain ontology instead of a user-centered one, generating such mappings may be a hard task as, for example, discussed in (Miller, Haas and Hernández, 2000). 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.

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*),

```
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 != 1 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_root-children(subSet)) then return false;

5. return true;
```

Figure 6. Generating (i+1)-sized combinations

the function *gen_comb_by_SS* (see Figure 6) 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.ii.A) fulfilling the necessary conditions (see Section 4.1):

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 verifies them as follows:

(a) If an *i*-sized subset of the *(i+1)*-sized set is in the *IDs* set (Step 4.a) 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 4.b), the *(i+1)*-sized combination can still be minimal and some of its 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 root-children (Step 4.b.i), the *(i+1)*-sized combination must be refuted, since our algorithm checks all root-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 independent, which means 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 the sets {BeginningDate, MinDuration} and {BeginningDate, Money} to get {BeginningDate, MinDuration, Money}. Then, function *check_Subsets* will generate its three 2-sized subsets: {BeginningDate, MinDuration}, {BeginningDate, Money} and also {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 root-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 root-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 the node {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. Algorithm Correctness

Our algorithm is sound and complete with regard to the domain ontology and data, as it generates sets from knowledge captured in the ontology and verifies them with the data. In this section we show that we generate all the sets of the searching space that fulfill the three necessary conditions, and that we do not generate any set that does not fulfill them. Thus, our proofs of soundness and completeness rely on the three necessary conditions discussed and justified in section 4.1. Note that, without considering our pruning rules, our algorithm would generate all the possible combinations of the searching space. For this reason, we just need to show that our pruning rules do not miss any ID nor generate combinations not being ID. In other words, in our algorithm we generate all the combinations of the searching space that (i) all its subsets are not IDs (i.e., all its SS-ancestors are not IDs), (ii) its components are pairwise orthogonal (i.e., we cannot find two elements in the ID such that one is in the FD-tree of the other) and (iii) its *intermediate sets* are, indeed, IDs (i.e., all its FD-ancestors are IDs). If we are able to generate all the sets meeting the necessary conditions and no other sets, it is easy to see that by verifying them with data (see section 5.1) our algorithm is still sound and complete.

We proved it by induction on the size of the combinations, and divided our proof in two parts. The first part shows that our algorithm is sound and complete for root-children (i.e., those combinations only consisting of concepts that are direct descendants of the root of the FD-tree). Then, we show that it is also sound and complete for *FD-descendants* (i.e., those combinations that contain, at least one concept whose distance to the root of the FD-tree is greater than 1). In each case, for the sake of understandability, we only provide proof of iterations

1 and 2. To find the proof of the induction step, which is a generalization from the second iteration, please see (Romero, 2010).

6.1. Root-children

In this section we show that we generate all the root-children satisfying the three necessary conditions (completeness) and no other root-child (soundness).

- **First Iteration:**
  `soundness:` In the first iteration, each concept functionally determined by A (not considering transitivity) is considered a set by itself, and they trivially satisfy Props. 4.3 and 4.4.
  `completeness:` In this iteration, every one of these concepts is considered a feasible ID, as they are the smallest set to be verified as an ID (see figure 5, step 4).
- **Second Iteration:**
  `soundness:` 2-sized root-children are generated by combining all the 1-sized candidate sets found in the previous iteration. By definition, those 1-sized sets in the candidate sets have been refuted as IDs (i.e., fulfilling Prop. 4.3), and only those 2-sized sets consisting of orthogonal concepts are generated (i.e., fulfilling Prop. 4.4. See figure 6, step 2.b.ii).
  `completeness:` We generate all the 2-sized combinations between concepts except for (i) those consisting of a concept being an ID, and (ii) those consisting of two concepts such that one is functionally determined by the other.

Finally, note that we have not justified Prop. 4.5 in this section. This is because root-children do not have intermediate sets by definition.

6.2. FD-descendants

In this section we show that we generate all the FD-descendants satisfying the three necessary conditions (completeness) and no other combination (soundness).

- **First Iteration:**

  `soundness:` 1-sized FD-descendants are generated following FDs from combinations verified as IDs and thus, fulfilling Prop. 4.5. These new generated combinations are of size 1 and they also fulfill Props. 4.3 and 4.4.
  `completeness:` All the 1-sized FD-descendants are generated iteratively following FD-edges (i.e., it follows the FD-edges until it finds FD-descendants that are not an ID). Therefore, those FD-descendants not explored will not be a feasible ID since they do not fulfill Prop. 4.5.

- **Second Iteration:**

  `soundness:` 2-sized FD-descendants can be generated from *gen_ comb_ by_ SS* function (see figure 6) or by following FD-edges from 2-sized combinations verified as IDs. In the first case, they fulfill the necessary conditions by the same proof introduced for 2-sized root-children. In the second case, it trivially guarantees Prop. 4.5. Furthermore, this also guarantees Props. 4.3 and 4.4. Let W, Z be two 2-sized sets of concepts such that W is related

to Z by FD-edges (maybe transitively) (i.e. Z was generated by following FD-edges). By definition of the FD-edges (see section 5) there is a concept B such that $B \subset Z$ and $B \subset W$, and two concepts A, A_1 such that $A \rightarrow A_1$, and $A \subset W$, and $A_1 \subset Z$. Z satisfies Prop.4.3 because if W is generated, B is not an ID (since it is in the candidate sets), and if A_1 is an ID, then A is an ID as well. Thus, A would have been verified as an ID in the first iteration, and, by following FD-edges, A_1 as well (so that, A_1 would be in the *IDs* set and Z would not be generated). Prop. 4.4 is guaranteed because 2-sized sets are generated if concepts on it are orthogonal (see figure 6; step 2.b.ii).

`completeness:` We show that our algorithm finds all the 2-sized FD-descendants by contradiction. Let Z be a 2-sized FD-descendant not found by our algorithm. If we have not been able to generate Z, at least one of its concepts (i.e., A_1) must not be among the 1-sized candidates. Therefore, A_1 is not a root-child (as every concept being a descendant of the root in the FD-tree will be present in the candidates set or in the IDs set; see second iteration in previous section). By definition of our searching space, a 2-sized root-child W exists such that W is an intermediate set of Z (see section 5). In other words, W is related to Z by FD-edges (maybe transitively). Thus, by Prop. 4.5, if Z is an ID then W is also an ID. Since our algorithm is exhaustive for root-children combinations, W would have been generated, and Z as well by following FDs.

7. Actually a Generic Algorithm

Despite it looks like a specific algorithm to discover IDs, it can actually be used to speed up the checking of any property fulfilling the three necessary conditions described in section 4.1. For example, in the DW area, most methods that automate the dimensional design process focus on identifying the dimensional knowledge contained in the sources. It is well known that these approaches tend to generate too many results, unnecessarily overwhelming the users with blindly generated combinations whose meaning has not been analyzed in advance. For example, (Phipps and Davis, 2002) identifies 6 facts in the TPC-H benchmark and proposes (out of 61 attributes in the source) 51, 13, 31, 35, 21, and 17 dimensional concepts to analyze each one of them; with our approach in (Romero and Abelló, 2010) we found for the 6 more promising facts (out of 170 properties and 65 concepts in the exemplifying ontology) 81, 78, 81, 80, 74, and 74 dimensional concepts. Clearly, these are too many results to be shown to the user for such small schemas.

In the data mining area, incorporating prior knowledge about the attributes has long been known to profoundly influence the effectiveness of analysis. This has been demonstrated by many authors, like (Kira and Rendell, 1992) and (Kohavi and John, 1997), using various heuristics that filter the attributes to be used on building a model. Thus, we could just redefine the acronym ID so that it stands for statistically *interesting analysis dimension*. In this way, using our algorithm we should be able to filter and prioritize the dimensional concepts found in the sources for a given fact, so that the user can focus on these to decide and define his/her requirements for an OLAP application. All we need now is to redefine the test we have to perform over data in this case (the rest of the algorithm remains exactly the same).

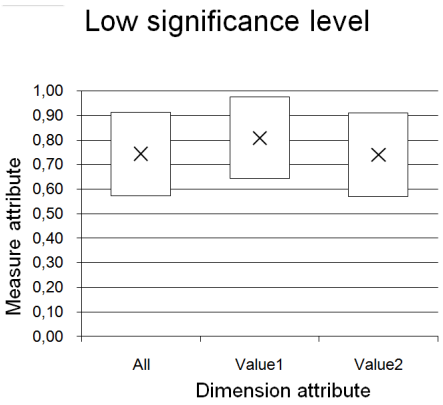

Figure 7. ANOVA graphical example

Shannon entropy quantifies the information contained in a variable. In this sense, it is used in (Yang, Procopic and Srivastava, 2009) to choose the most important tables in a relational schema. In our case, we do not want to decide the absolute importance of a concept, but its importance when it partitions a measure. Thus, we should choose the IDs[5] based on the gain of entropy on partitioning each measure (this would show how much information we gain on analyzing the measures from a given point of view). Unfortunately, entropy only applies to discrete variables and measures in a DW use to be continuous. Thus, we propose to measure the interest of partitioning the values of a concept by analyzing the consequences in the variance.

In (Wonnacott and Wonnacott, 1990) we can see how to perform an analysis of variance (ANOVA). This is a test designed to decide whether the difference in the means of several samplings are due to differences in the populations or can be reasonably attributed to chance fluctuations alone. Figures 7 and 8 sketch the idea behind ANOVA test. Figure 7 shows a measure partitioned by a potential dimensional attribute in two sets, while Figure 8 shows it partitioned by another potential analysis dimension in six sets. The "×" marks the average for either the total (i.e. "All") or each partition, and the rectangle containing it displays the interval defined by the standard deviation. Even though in both cases the average of some subsets differ from the total, in Figure 7 it is statistically plausible being just by chance, while in Figure 8, within a 99% confidence interval, the difference in the distribution of the measure in each category really comes from the partitioning concept.

Thus, in our case, the hypothesis of "no difference" in the population of the different subsets is the *null hypothesis*. If this is rejected in our statistical analysis with a given confidence level, we will propose this concept (or set of concepts) generating the partition as an ***Interesting Dimension* (ID)** to the user. Nevertheless, regardless of the statistical conclusions, it will still be up to the user to decide whether a given ID is truly interesting for the organization or not. Note that not only individual concepts, but also sets of concepts could give rise to a single ID.

[5] We recall the reader that, in this section, IDs stands for *Interesting Dimensions*.

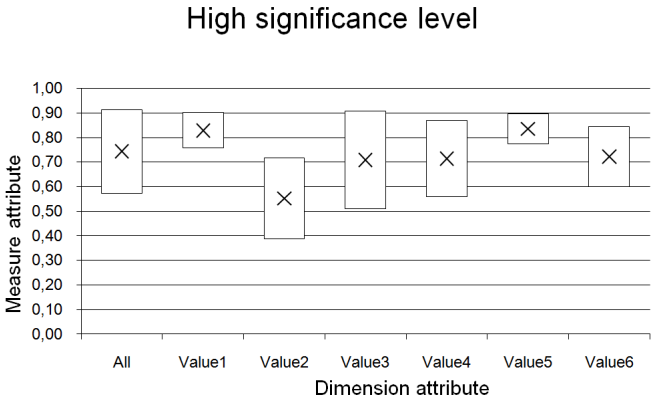

Figure 8. ANOVA graphical example

Again our search space are all combinations of concepts in the ontology, but since the same properties in Section 4.1 apply again to this new relationship, we can reuse the same algorithm. All we need is to adapt the function that performs the final checking over data.

7.1. The *interesting* Function

This function is called, instead of "determines" (Section 5.1), when the three necessary conditions are guaranteed (i.e., we have identified a feasible ID). Then, we verify whether this combination of dimensional concepts is interesting to analyze the measure by querying data.

Similar to Proposition 5.1, prior to perform the statistical analysis, we may first introduce a final pruning rule over the catalog:

Proposition 7.1. A concept with too many instances is not interesting as a dimension.

Statisticians consider that useful categorical variables must have, at most, some tens of values. Thus, non-numerical concepts with more than, for example, one hundred values may be discarded in the experiments, while numerical concepts with more than one hundred values should be discretized to reduce the number of values (when used as a dimensional concept), see for example (Bondu, Boullé and Lemaire, 2010). Those combinations satisfying this rule are still candidates to be an ID, and we verify it by performing a one-way ANOVA test with the following query:

```
SELECT (SUM(gr.s)/(#distinct -1))/(SUM(POWER(A-grAvg,2))/(#tuples -#distinct))
       AS fFisher
FROM t, (SELECT attrSet AS id,
                avg(A) AS grAvg,
                POWER(AVG(A)-(SELECT avg(A) FROM t),2) AS s
         FROM t
         WHERE joinConds
         GROUP BY attrSet) gr
WHERE attrSet=gr.id;
```

Where *attrSet* are the concepts conforming the feasible ID to be verified,

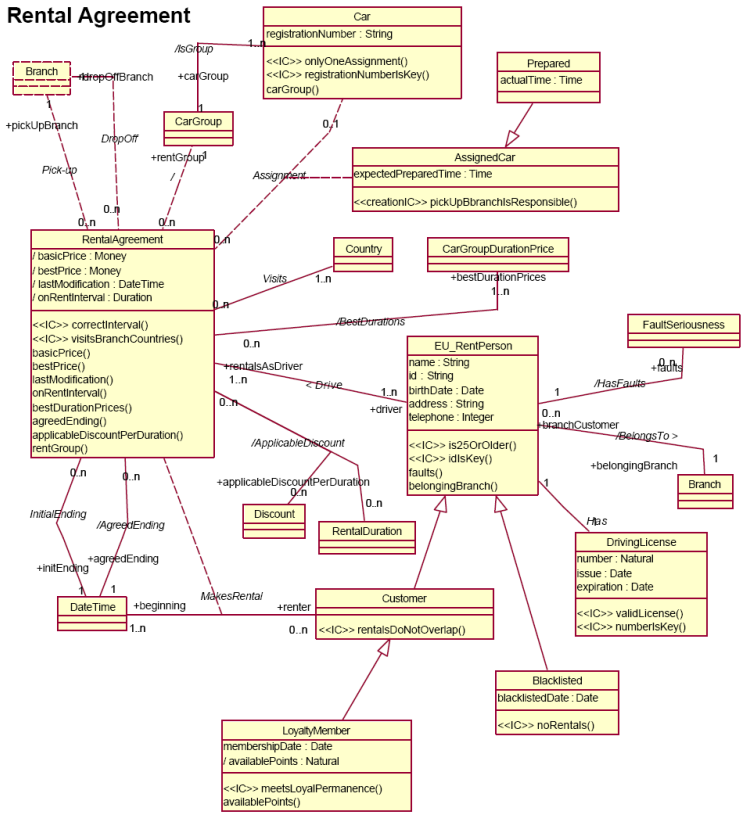

Figure 9. Main concepts in the Rental Agreement ontology

t the table or tables containing the corresponding attributes (join conditions should be added if necessary), $\#distinct$ is the number of different values for $attrSet$ and $\#tuples$ the number of tuples in the fact table. Then, the credibility of the null hypothesis is obtained by placing the result of this query in a Fisher distribution with the corresponding degrees of freedom. For example, for an $fFisher$ value obtained in the query of 2.19, given 8 degrees of freedom (i.e. $\#distinct$=9) for numerator and 379 (i.e. $\#tuples$=388) for denominator results in a 97% of confidence, which should be interpreted as the confidence of the current combination of concepts being statistically interesting.

8. Case Study

In this section, we introduce results got after carrying out our algorithm over the EU-Car Rental case study (Frías et al., 2003). This ontology refers to a car renting domain [6] and it has 65 concepts and 170 relationships (Figure 9 shows the main ones).

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

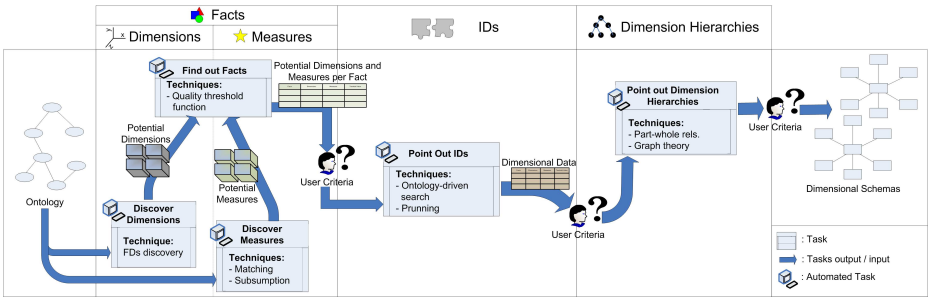

Figure 10. Process overview

Specifically, we used the implementation of the method in (Romero and Abelló, 2010) and included this work interleaved in the process. Figure 10 depicts a schematic overview. Throughout three well-differentiated tasks, it identifies concepts likely to play multidimensional roles and therefore, likely to give rise to dimensional schemas. 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. Identifying facts automatically is a hard task, and most methods rely on heuristics such as table cardinalities or numerical attributes. We assumed that an ontology concept is likely to play a fact role if it has as many measures as possible and it can be analyzed from as many different perspectives as possible. A fact with these characteristics may not be of interest to the user (this will be considered later in our approach), but objectively, it will provide many different measures to be analyzed from many different perspectives. This task, therefore, is divided in two main subtasks; (1) discover potential dimensional concepts and (2) point out potential measures. Now, for each ontology concept we can estimate its likelihood of being a fact. In general, those concepts with most potential dimensional concepts and measures are good candidates, but we should weight each input according to our preferences. In our approach we define f as a function that, given the number of dimensional concepts and measures of a concept `c`, it evaluates it as a promising fact. This quality function will prune those concepts below a given threshold. Potential facts not pruned are ranked according to its f value and presented to the user as in the following table.

Concept	**#Dimensional Concepts**	**#Potential Measures**	$f = \#D + 2\#M$
LateReturn	78	5	88
DamageCost	81	3	87
Prepared	81	3	87
AssignedCar	80	3	86
PaidWithPointsRental	74	4	82
ClosedRental	74	4	82
EarlyReturn	74	4	82

Lets suppose now that the user selects `ClosedRental`, which is a subclass of `RentalAgreement` representing those agreements that have been already closed (lets say "historic rental agreements"). The second task uses the algorithms in

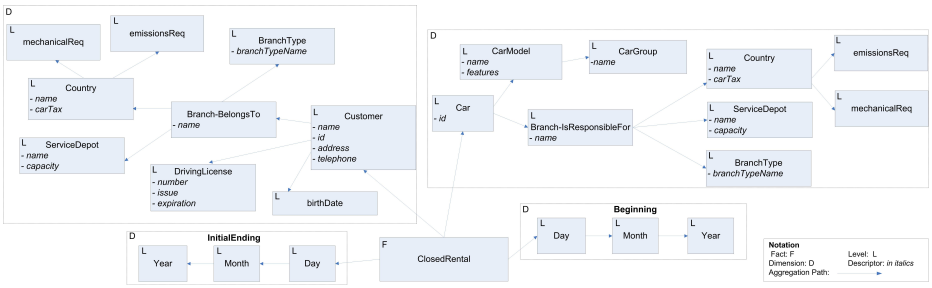

Figure 11. Dimensional schema

this paper to show to the user the different potential IDs of the chosen fact. We get its FDs (from knowledge asserted in the ontology) and later, by means of the algorithm introduced in Section 5, its *feasible IDs*. In this case, the algorithm generates only 149 *feasible IDs* to be tested with data, out of the more than 10^{21} combinations of the 74 dimensional concepts. Afterwards, the user will validate which IDs are meaningful to the business. Among the results, the user would find, IDs like {`Beginning`,`Customer`}, {`Beginning`,`Car`}, {`InitialEnding`,`Customer`}, {`InitialEnding`,`Car`}, {`AgreedEnding`,`Customer`}, and {`AgreedEnding`,`Car`}.

With this, the system would generate metadata to inform the user of the different dimensional spaces where data can be placed and avoid sparse cubes (since, by definition, IDs are minimal sets). Moreover, it would allow to identify analysis dimensions for the fact (since they are precisely the components of the IDs). Lets suppose, now, that the user selected the first four IDs out of those mentioned. The last task gives rise to dimension hierarchies for these concepts. In other words, we aim to identify relevant aggregation paths looking for typical part-whole relationships. In this step, graphs giving shape to each dimension hierarchy are built up. The user may again tune them up to his/her needs.

Finally, Figure 11 shows the dimensional schema obtained for `ClosedRental`. The analysis dimensions found show the `Car` and the `Customer` involved in the agreement, together with the time of the agreement and the pickup time of the car. We claim that to derive all the multidimensional knowledge contained in the ontology and let the user just filter results got according to his/her requirements is much easier than doing it completely manually without any assistance.

If we apply the second task in the process to each and every concept in the ontology, we have 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), the average degree of these nodes was 6.67 (i.e., the average number of combinations we start with in the first iteration of our algorithm) and the average depth was 2.91 (with a minimum of 1 and a maximum of 5). As a result, when computing IDs we get an average of 156.53 *feasible IDs* per concept (i.e., we would query the database only 156.53 times per concept; in front of the 3,817,550,246 times if we would have generated all the combinations in the searching space, i.e., $2^{31.83}$). Figure 12 puts these numbers in a bar chart of logarithmic scale.

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

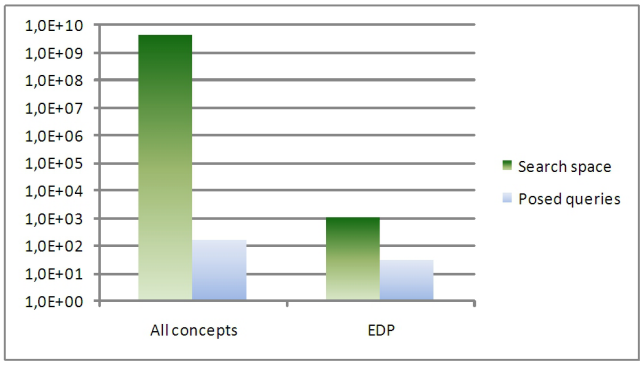

Figure 12. Summary of results

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 in the ontology 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 (Stanford Center for Biomedical Informatics Research, 2009)).

We do not compare our approach against those other methods testing the existence of an FD, because ours would be a pre-process for the others. We would just need to substitute our naive SQL query in Section 5.1 by any other smarter method that allows to determine whether or not a set of attributes are an ID of a given class of objects (as demonstrated in Section 7.1). Our improvement is on reducing the amount of times we need to execute those other methods by using knowledge not at the data level, but at the conceptual level.

9. Conclusions

We have proposed an algorithm for finding compound IDs (i.e., compound 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. Even if an ontology is not available, it would be easier to generate it (which can be automatized to some extent, as shown in the literature) than testing only over data. We have shown soundness and completeness proofs with regard to the knowledge captured in the domain ontology and the data, and presented the feasibility of our method by means of the statistics raised by the implementation of our algorithm over a case study. Despite we focus on the usage of the algorithm in data warehouse design problems, it can also be used in other areas such as data cleansing, integration, or

any other reengineering process where would be necessary to discover meaningful IDs. Moreover, we also explained how this same algorithm can be adapted to other different problems where the three properties we use still hold (intuitively, when defining another characteristic sharing the properties of the FDs). We exemplified it by testing whether the combination of a set of concepts is statistically interesting to analyze a given measure (where interesting is defined in terms of passing ANOVA tests).

As future work, we plan to adapt the algorithm to look for keys for all the domain concepts at the same time instead of performing this task one by one.

Acknowledgments

This work has been partly supported by the Ministerio de Ciencia e Innovación under project TIN2008-03863. We would also like to thank Tomas Aluja for his help on statistical issues, Maria C. Keet for her insights and Joan Marc Montesó for the implementation of the algorithm.

References

Abelló, A. and Romero, O. (2010), Using Ontologies to Discover Fact IDs, *in* 'ACM 13th Int. Workshop on Data Warehousing and OLAP', ACM, p. To appear.

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

Abiteboul, S., Hull, R. and Vianu, V. (1995), *Foundations of Databases*, Addison-Wesley.

Artale, A., Calvanese, D., Kontchakov, R., Ryzhikov, V. and Zakharyaschev, M. (2007), Reasoning over Extended ER Models, *in* 'Proc. of 26th Int. Conf. on Conceptual Modeling', Vol. 4801 of *Lecture Notes in Computer Science*, Springer, pp. 277–292.

Assawamekin, N., Sunetnanta, T. and Pluempitiwiriyawej, C. (2010), 'Ontology-based multiperspective requirements traceability framework', *Knowledge and Information Systems* **25**(3), 493–522.

Berardi, G., Calvanese, D. and Giacomo, D. (2005), 'Reasoning on UML Class Diagrams', *Artificial Intelligence* **168**(1-2), 70–118.

Berlanga, R., Jiménez-Ruiz, E., Nebot, V. and Sanz, I. (2010), 'Faeton: Form analysis and extraction tool for ontology construction', *Journal of Computer Applications in Technology* **39**(4), 224–233.

Berners-Lee, T., Hendler, J. and Lassila, O. (2001), 'The Semantic Web', *Scientific American* **284**(5).

Bondu, A., Boullé, M. and Lemaire, V. (2010), 'A non-parametric semi-supervised discretization method', *Knowledge and Information Systems* **24**(1), 35–57.

C. Monash (2008), 'The 1-Petabyte Barrier is Crumbling'. http://www.networkworld.com/community/node/31439 (last access 22/11/2010).

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

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

Codd, E. F. (1990), *The Relational Model for Database Management, Version 2*, Addison-Wesley.

Dánger, R. and Berlanga, R. (2009), 'Generating complex ontology instances from documents', *J. Algorithms* **64**(1), 16–30.

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

Francisco, V., Gervás, P. and Peinado, F. (2010), 'Ontological reasoning for improving the treatment of emotions in text', *Knowledge and Information Systems* **25**(3), 421–443.

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

Gaševic, D., Djuric, D. and Devedžic, V. (2007), 'MDA-based Automatic OWL Ontology Development', *Int. Journal on Software Tools for Technology Transfer* **9**(2), 103–117.

Giorgini, P., Rizzi, S. and Garzetti, M. (2005), Goal-oriented Requirement Analysis for Data Warehouse Design, *in* 'Proc. of 8th Int. Workshop on Data Warehousing and OLAP', ACM Press, pp. 47–56.

Golfarelli, M., Maio, D. and Rizzi, S. (1998), 'The Dimensional Fact Model: A Conceptual Model for Data Warehouses.', *Int. Journal of Cooperative Information Systems* **7**(2-3), 215–247.

Golfarelli, M. and Rizzi, S. (2009), *Data Warehouse Design*, McGraw-Hill.

Hainaut, J., Chandelon, M., Tonneau, C. and Joris, M. (1993), Contribution to a Theory of Database Reverse Engineering, *in* 'Proc. of the 1st Working Conf. on Reverse Engineering', IEEE, pp. 161–170.

Halpin, T. and Morgan, T. (2008), *Information Modeling and Relational Databases*, Morgan Kauffman.

Huhtala, Y., Kärkkäinen, J., Porkka, P. and Toivonen, H. (1999), 'Tane: An efficient algorithm for discovering functional and approximate dependencies', *Comput. J.* **42**(2), 100–111.

Jensen, M. R., Holmgren, T. and Pedersen, T. B. (2004), Discovering Multidimensional Structure in Relational Data., *in* '6th Int. Conf. on Data Warehousing and Knowledge Discovery', Springer, pp. 138–148.

Kimball, R., Reeves, L., Thornthwaite, W. and Ross, M. (1998), *The Data Warehouse Lifecycle Toolkit: Expert Methods for Designing, Developing and Deploying Data Warehouses*, John Wiley & Sons, Inc.

King, R. S. and Legendre, J. J. (2003), 'Discovery of functional and approximate functional dependencies in relational databases', *JAMDS* **7**(1), 49–59.

Kira, K. and Rendell, L. A. (1992), A practical approach to feature selection, *in* 'Machine Learning', pp. 249–256.

Kohavi, R. and John, G. H. (1997), 'Wrappers for feature subset selection', *Artif. Intell.* **97**(1-2), 273–324.

Lenzerini, M. (2002), Data Integration: A Theoretical Perspective, *in* 'Proc. of 21th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems', ACM, pp. 233–246.

Lopes, S., Petit, J.-M. and Lakhal, L. (2000), Efficient discovery of functional dependencies and armstrong relations, *in* '7th Int. Conf. on Extending Database Technology, EDBT'00', Vol. 1777 of *Lecture Notes in Computer Science*, Springer, pp. 350–364.

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

Miller, R. J., Haas, L. M. and Hernández, M. A. (2000), Schema mapping as query discovery, *in* 'Proceedings of 26th International Conference on Very Large Data Bases, VLDB'00', Morgan Kaufmann, pp. 77–88.

Nebot, V. and Berlanga, R. (2009), Building tailored ontologies from very large knowledge resources, *in* 'Proceedings of the 11th International Conference on Enterprise Information Systems, ICEIS'09', pp. 144–151.

Novelli, N. and Cicchetti, R. (2001), Fun: An efficient algorithm for mining functional and embedded dependencies, *in* '8th International Conference on Database Theory, ICDT'01', Vol. 1973 of *Lecture Notes in Computer Science*, Springer, pp. 189–203.

Olivé, A. (2004), On the Role of Conceptual Schemas in Information Systems Development, *in* '9th Int. Conf. on Reliable Software Technologies', Springer.

OMG (2010), 'Unified Modeling Language (UML), Version 2.3'. http://www.omg.org (last access 20/10/10).

Phipps, C. and Davis, K. C. (2002), Automating Data Warehouse Conceptual Schema Design and Evaluation., *in* '4th Int. Workshop on Design and Management of Data Warehouses', Vol. 58, CEUR-WS.org, pp. 23–32.

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

Ramakrishnan, R. and Gehrke, J. (2003), *Database Management Systems*, McGraw Hill.

Romero, O. (2010), Automating the Multidimensional Design of Data Ware-

houses, PhD thesis, Universitat Politècnica de Catalunya, Barcelona, Spain. http://www.tdx.cat/handle/10803/6670.

Romero, O. and Abelló, A. (2010), 'A Framework for Multidimensional Design of Data Warehouses from Ontologies', *Data & Knowledge Engineering* **69**(11), 1138 – 1157.

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

Sismanis, Y., Brown, P., Haas, P. J. and Reinwald, B. (2006), GORDIAN: Efficient and Scalable Discovery of Composite Keys, *in* '32nd Int. Conf. on Very Large Data Bases', ACM, pp. 691–702.

Skoutas, D. and Simitsis, A. (2007), 'Ontology-Based Conceptual Design of ETL Processes for Both Structured and Semi-Structured Data', *International Journal on Semantic Web and Information Systems* **3**(4), 1–24.

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

Stanford Center for Biomedical Informatics Research (2009), 'Protégé-OWL API (last access 17/12/2009)'. protege.stanford.edu/plugins/owl/api.

T., H. B. K. and Zhao, Y. (2004), 'Automated Elicitation of Functional Dependencies from Source Codes of Database Transactions', *Information & Software Technology* **46**(2), 109–117.

W3C (2009), 'OWL Web Ontology Language Overview'. http://www.w3.org/TR/owl-features/ (last access 17/12/2009).

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

Wonnacott, T. H. and Wonnacott, R. J. (1990), *Introductory Statistics*, Wiley&Sons.

Yang, X., Procopic, C. M. and Srivastava, D. (2009), Summarizing Relational Databases, *in* 'Int. Conf. on Very Large Databases (VLDB)', ACM, pp. 634–645.

Yao, H. and Hamilton, H. J. (2008), 'Mining functional dependencies from data', *Data Min. Knowl. Discov.* **16**(2), 197–219.

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

Author Biographies

Alberto Abelló is tenure-track 2 professor at Universitat Politècnica de Catalunya (Polytechnical University of Catalonia). He received his PhD from UPC in 2002. He is also a member of the MPI research group (Grup de recerca en Modelització i Processament d'Informació) at the same university, specializing in software engineering, databases and information systems. His research interests are databases, database design, data warehousing, OLAP tools, ontologies and reasoning. He is the author of articles and papers presented and published in national and international conferences and journals on these subjects (e.g., Information Systems, Data and Knowledge Engineering (DKE), International Journal of Data Warehousing and Mining, Conference on Advanced Information Systems Engineering (CAiSE), International Conference on Database and Expert Systems Applications (DEXA), International Conference on Data Warehousing and Knowledge Discovery (DaWaK) and International Workshop on Data Warehousing and OLAP (DOLAP). He has also served in the program committee of DaWaK and DOLAP.

Oscar Romero is a tenure-track 1 lecturer at the Barcelona School of Informatics. He has an MSc and a PhD in computing from the Universitat Politècnica de Catalunya. He is also a member of the MPI research group at the same university, specialized in software engineering, databases and information systems. His research interests are data warehousing, OLAP tools, multidimensional modeling, the semantic web and ontologies. He is the author of papers presented and published in national andinternational conferences and journals on these subjects.