Starry Vault: Automating Multidimensional Modeling from Data Vaults

Matteo Golfarelli(✉), Simone Graziani, and Stefano Rizzi

DISI, University of Bologna, V.le Risorgimento 2, 40136 Bologna, Italy
{matteo.golfarelli,simone.graziani2,stefano.rizzi}@unibo.it

Abstract. The data vault model natively supports data and schema evolution, so it is often adopted to create operational data stores. However, it can hardly be directly used for OLAP querying. In this paper we propose an approach called *Starry Vault* for finding a multidimensional structure in data vaults. Starry Vault builds on the specific features of the data vault model to automate multidimensional modeling, and uses approximate functional dependencies to discover out of data the information necessary to infer the structure of multidimensional hierarchies. The manual intervention by the user is limited to some editing of the resulting multidimensional schemata, which makes the overall process simple and quick enough to be compatible with the situational analysis needs of a data scientist.

Keywords: Data vault · Data warehouse design · Multidimensional modeling

1 Introduction

Since their adoption as an enabling technology for information systems, one of the goal of databases has been to provide a unified, integrated, and consistent repository for *all* enterprise data; this repository should act has a hub for different activities such as process coordination, auditing, historical data storage, etc. Among the solutions devised in this direction we mention Master Data Management and ERPs in the area of operational systems; in the area of business intelligence, Operational Data Stores and, more recently, data lakes. Another solution that has been progressively gaining attention and diffusion since its official release in 2000 is the *data vault*, a practitioner-driven proposal for designing a database that provides long-term historical storage of data coming in from multiple sources. The main goals of the data vault can be summarized as (i) maximize resilience to change in the business environment when storing historical data; (ii) accommodate data regardless of their quality and of their conformity to standard and business rules; and (iii) enable parallel loading so that very large implementations can scale out without the need of major redesign. While

This work was partly supported by the EU-funded project TOREADOR (contract n. H2020-688797).

J. Pokorný et al. (Eds.): ADBIS 2016, LNCS 9809, pp. 137–151, 2016.
DOI: 10.1007/978-3-319-44039-2_10

the 1.0 version of the data vault was strictly relational, version 2.0 (released in 2015) relies on Hadoop-Hive for delivering scalability and performance at a big data level. However, in spite of its undeniable informative value, a data vault is not suitable for direct multidimensional querying both for performance reasons (it is not optimized for OLAP workloads) and because it is hardly supported by OLAP front-ends.

In this paper we propose an approach called *Starry Vault* aimed at finding a multidimensional structure in data vaults so that their data can be fed into a data warehouse (DW) for OLAP querying. On the one hand, our approach builds on the specific features of the data vault model to automate multidimensional modeling, on the other it uses approximate functional dependencies [7] to discover out of data the information necessary to infer the structure of multidimensional hierarchies. The Starry Vault approach is mainly aimed at being used at design time, to support a supply-driven design of a DW from a source data vault [18]. However, the manual intervention by the user is limited to some editing of the resulting multidimensional schemata, which makes the overall process simple and quick enough to be also compatible with the situational analysis needs typical of a data scientist.

2 Related Work

The data vault model has hardly been explored in the academic literature. Besides the official model specification [14], to the best of our knowledge only a couple of works were made: [11], which provides a conceptualization of the data vault physical model, and [13], which describes an approach for designing DWs where the data vault model is used instead of the standard star/snowflake schemata to physically implement the multidimensional model. On the other hand, there are evidences that the data vault can be used in agile design contexts [6], and some CASE tools generate DW schemata based on the data vault model (e.g., Quipu [16]).

The problem of how to support or even automate the design of DWs has been widely explored. In particular, in *supply-driven* approaches multidimensional modeling starts from an analysis of data sources—which is in line with the goal of this paper. The first approaches to supply-driven design date back to the late 90's [3,10,12,15] and propose algorithms that create multidimensional schemata starting from Entity-Relationship diagrams or relational schemata. The basic idea is that of following the functional dependencies (FDs) expressed in the source schema to build the multidimensional hierarchies. In the following years, there have been some attempts to obtain multidimensional schemata out of XML source data (e.g., [5]). In this case, the main problem is that some FDs are not intensionally expressed, so they must be checked extensionally, i.e., by properly querying the XML database at design time.

The main inspiration for our current work comes from the supply-driven approaches that use relational schemata as a source. However, these approaches cannot be smoothly reused in our case because (i) while in traditional (normalized) relational databases all FDs are made explicit, several FDs are normally

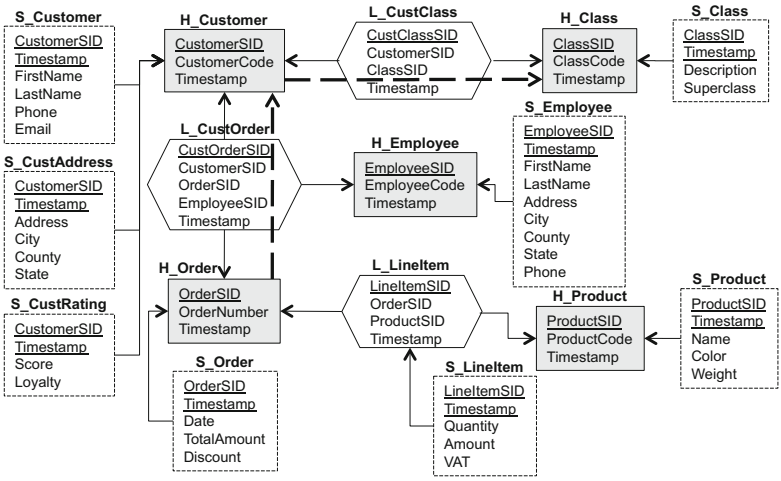

Fig. 1. A sale data vault. Grey boxes, hexagons, and dashed boxes represent hubs, links, and satellites, respectively; additional FDs are shown with thick dashed arrows

hidden in data vaults; (ii) the peculiar structure of data vaults, lets us make some specific assumptions which are not possible with traditional relational databases; (iii) while relational-based approaches do not use many-to-many relationships for design, these must always be considered when designing from data vaults. On the other hand, the idea of querying data vaults to establish the missing FDs is borrowed from the approaches using XML sources.

Among the works on supply-driven design of DWs, some also consider the problem of supporting the designer in detecting potential facts. For instance, in [15] all the entities with numeric fields are selected as candidate facts. Not only the presence of measures, but also table cardinality is considered to identify facts in [10], while in [12] all entities with a high number of many-to-one relationships are candidates to become facts. A model-driven approach to detect fact is proposed in [1], based on a heuristics that considers the cardinality and in-degree of each table, together with its ratio of numerical fields. Finally, in [17] potential facts are selected by searching specific topological patterns in source data. The criteria we use in this work for ranking candidate md-schemata are partially inspired and adapted from the ones mentioned above.

3 Data Vault Basics

The data vault model was conceived by Dan Linstedt in 1990 and then released in 2000 as a public domain modeling method [14]. Its basic goal is that of dealing with data and schema changes by separating the business keys (that are basically stable, because they uniquely identify a business entity) and the associations between them, from their descriptive attributes (that may change frequently). The data vault is based on three components [8]:

- *Hubs.* A hub is a table that models a core concept of business; each of its tuples corresponds to a single business object with a unique enterprise-wide key, and is timestamped with the moment that object was first loaded into the database. The primary key of a hub is always a surrogate key.
- *Links.* A link is a table that models a business relationship between hubs. To establish this relationship, a link includes foreign keys referencing the hubs/links involved. Like a hub, it has a surrogate as the primary key and it includes a load timestamp. To ensure that the schema can be easily evolved, all relationships are modeled as potentially many-to-many regardless of their actual multiplicity.
- *Satellites.* A satellite is a table that includes a set of attributes describing one hub or one link. Its primary key combines a foreign key that references the corresponding hub/link with a timestamp, so that multiple temporal version of attribute values can be stored.

Example 1. The simple data vault we will use as a working example models sale orders and is shown in Fig. 1 (adapted from [8]).

4 Formal Background

In this section we give a graph-based formalization of data vaults and multidimensional schemata, which will be respectively the input and output of our design algorithm.

Definition 1 (Data Vault Schema). *A* data vault schema *(briefly,* dv-schema*) is a directed graph* $\mathcal{V} = (T, F)$ *where* $T = T_H \cup T_L \cup T_S$ *and:*

1. T_H, T_L, *and* T_S *are, respectively, sets of hub, link, and satellite tables;*
2. *each arc* $\langle t, t' \rangle$ *in* F *represents an FD from a foreign key of table* t *to the primary key of table* t'*, which we will denote with* $t \rightarrow t'$ *to emphasize that one tuple of* t *determines one tuple of* t'*;*
3. $F \subseteq (T_S \times (T_H \cup T_L)) \cup (T_L \times T_H)$*;*
4. *exactly one arc exits from each satellite* $s \in T_S$ *(entering a hub or a link);*
5. *at least two arcs exit from each link.*

Given point (3) of Definition 1, all FDs explicitly modeled in a dv-schema take either form $s \rightarrow h$, $s \rightarrow l$, or $l \rightarrow h$. Each hub in $h \in T_H$ has one business key, denoted $BusKey(h)$. Each satellite s has a set of business attributes, $BusAttr(s)$; for each hub or link t, we denote with $BusAttr(t)$ the union of the sets of business attributes included in all satellites s such that $s \rightarrow t$.

Example 2. With reference to the sale data vault in Fig. 1, it is $T_H = \{$H_Customer, H_Order, H_Employee, H_Class, H_Product$\}$, $T_L = \{$L_CustClass, L_CustOrder, L_LineItem$\}$, and $T_S = \{$S_Customer, S_CustAddress, S_CustRating, $\ldots\}$. An example of arc is $\langle$L_CustClass, H_Class$\rangle$, which corresponds to the inter-table FD L_CustClass $\rightarrow$ H_Class. Finally, it is $BusKey($H_Customer$)$ = CustomerCode and $BusAttr($H_Customer$)$ = {FirstName, LastName, Phone, Email, Address, City, County, State, Score, Loyalty}.

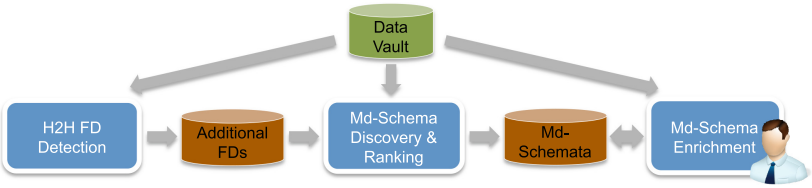

Fig. 2. Process architecture of the Starry Vault approach

Definition 2 (Multidimensional Schema). *A* multidimensional schema *(or* md-schema*) is a directed acyclic graph* $\mathcal{M} = (A, E)$ *where each node in* A *is an attribute, each arc in* E *is an FD involving two attributes, and there exists one node* $f \in A$*, called* fact*, such that each other node in* A *can be reached from* f *through a directed path (which implies that* f *has no entering arcs). The set of direct children of* f *is partitioned into a set of dimensions,* D*, and a set of measures,* M*. All measures in* M *are leaves of* $\mathcal{M}$*. For each dimension* $d \in D$*, the subgraph of* $\mathcal{M}$ *that can be reached from* d *is called a* hierarchy*.*

5 The Starry Vault Approach

A functional overview of the approach we use to obtain an md-schema out of a source dv-schema is sketched in Fig. 2; three processes are included:

1. *Hub-To-Hub FD Detection.* This process aims at detecting additional FDs not explicitly modeled in the dv-schema, in particular those between two or more hubs connected by a link, by querying the source data vault.
2. *Md-Schema Discovery and Ranking.* A set of candidate facts is heuristically determined; for each of them, a draft md-schema is built based on both the FDs explicitly modeled in the dv-schema and those detected by process (1). The md-schemata obtained are then heuristically ranked based on how comprehensive they are from the intensional and extensional points of view.
3. *Md-Schema Enrichment.* The user selects one or more draft md-schemata, then edits and enriches them based on her knowledge of the application domain. To further improve the quality of the md-schemata, additional FDs hidden in satellites can be discovered by querying the source data vault.

5.1 Hub-To-Hub FD Detection

In a dv-schema each relationship between two or more hubs is modeled through a link that contains the foreign keys referencing the connected hubs. As already mentioned, this implies that all relationships are modeled as if they were many-to-many, so it is not possible to determine if there are any FDs between two hubs (i.e., if a relationship is really many-to-many or is actually many-to-one) based on the dv-schema alone. For instance, looking at Fig. 1 it is impossible to

say if the binary relationship between customer and classes is many-to-many or, more realistically in this case, many-to-one.

Things get even more complex with n-ary relationships, like the one expressed by L_CustOrder that features three branches. Indeed, in this case there are different possibilities:

1. The relationship between the hubs involved really has many-to-many multiplicity in all directions. In particular, in case of the L_CustOrder link, this would mean that one order can be made by several customers with the support of several employees.
2. The relationship has many-to-one multiplicity from one branch towards the others. In our example, this happens if one order is always made by one customer with the support of one employee.
3. There are mixed multiplicities from the same branch. For instance, this is the case if one order is always made by one customer with the support of several employees.

Note that, while in a standard relational schema only case (1) corresponds to a good design practice for normalization reasons (in the other cases the n-ary relationship should be substituted by $n-1$ binary relationships, each with its multiplicity), within a dv-schema all three cases are considered equally good for the sake of maintainability.

To disambiguate relationship multiplicities in all cases above and detect FDs with reasonable confidence, we must resort to the data stored in the source data vault. Clearly, there is a chance that an FD holds for the specific data stored at design time but does not hold in general in the application domain, which means that it will probably be contradicted in the future when new data will be added. Fortunately, since data vaults usually host great amounts of data, these can realistically be considered to be representative of the application domain. More probably, the data will be affected by noise in the form of errors (e.g., spelling errors) that "hide" an existing FD. The tool we use to cope with this issue are *approximate functional dependencies* (AFDs) [7], i.e., FDs that "almost hold", which normally arise when there is a natural FD between attributes but data are dirty or present exceptions. Given AFD $a \rightsquigarrow b$, where a and b are attributes, one way to define its approximation $e(a \rightsquigarrow b)$ is to count the minimum number of distinct values of ab that must be removed to enforce $a \rightarrow b$. We will then consider $a \rightsquigarrow b$ to hold if $e(a \rightsquigarrow b) < \epsilon$, where ϵ is a threshold.

The approach we adopt to detect AFDs is an adaptation of the well-known Tane algorithm [7]. Given a table r with schema R, Tane computes all the valid AFDs $X \rightsquigarrow a$ with $X \subseteq R$ and $a \in R \setminus X$ by relying on a level-wise (small-to-big) enumeration strategy to navigate the search space of all possible subsets of R (i.e., the containment lattice). Though Tane applies a set of pruning rules to avoid computing/returning trivial and non-minimal dependencies, its complexity remains exponential due to the number of candidate attribute sets X. Specifically, the worst-case complexity of Tane is $O(|r| + |R|^{2.5})2^{|R|})$, where $|r|$ is the cardinality of table r and $|R|$ is its number of attributes. Noticeably, since our goal here is to build hierarchies, we can restrict our search to simple AFDs

($|X| = 1$). In the remainder of this section we describe an original enumeration strategy that works for simple AFDs and cuts the complexity of TANE down to $O(|r| \cdot |R^2|)$ in the worst case and to $O(|r| \cdot |R|)$ in the best one.

Let us start by considering "traditional" FDs. Given schema R, the set of candidate FDs $a \rightarrow b$, with $a, b \in R$, can be represented using an $|R| \times |R|$ matrix Z whose rows and columns represent left- and right-hand sides of FDs, respectively, so that $Z[a, b]$ corresponds to $a \rightarrow b$. If FD $a \rightarrow b$ is found to hold on the stored data, cell $Z[a, b]$ is set to true, otherwise it is set to false. A naive approach to fill Z would check each single cell, i.e., each possible simple FD by accessing data; actually, most checks can be avoided by orderly exploring the cells of Z. Our exploration strategy requires the rows and columns of Z to be ordered by descending cardinality of the corresponding attribute domain. Given the ordered matrix, we initially note that only the cells over the diagonal must be checked since (i) the cells on the diagonal correspond to trivial FDs like $a \rightarrow a$, and (ii) the cells below the diagonal correspond to unfeasible FDs like $b \rightarrow a$ with $|b| < |a|$. Among the cells above the diagonal of Z, we can avoid checking those corresponding to transitive FDs by applying the following exploration strategy:

- *Rule 1*: First check the (unchecked) cells $Z[a, b]$ such that $|b|$ is maximum and, among them, give priority to the one with minimum $|a|$.
- *Rule 2*: If the FD corresponding to $Z[b, c]$ is found to be true, set to true all the FDs corresponding to cells $Z[*, c]$ such that $Z[*, b]$ holds.

To understand why Rules 1 and 2 avoid checking transitive FDs, consider FDs $a \rightarrow b$ and $b \rightarrow c$, which transitively imply $a \rightarrow c$. Then it must be $|c| \leq |b| \leq |a|$, so due to Rule 1 the check of $a \rightarrow c$ is scheduled after those of $a \rightarrow b$ and $b \rightarrow c$. But since $b \rightarrow c$ holds, Rule 2 sets $a \rightarrow c$ to true before it is checked.

According to the previous enumeration rule, the number of candidate FDs that must be verified depends, given the number of attributes, on the number of transitive FDs in R. The worst case arises when no transitive FDs hold between the attributes in R, because all the cells in the upper-right half of Z (i.e., $|R| \times (|R| - 1)/2$ cells) must be checked. The best case takes place when the attributes of R are involved into a linear hierarchy, because the number of checks drops to $|R| - 1$. Considering that the complexity of TANE is determined by its enumeration strategy and that TANE checks the FDs in linear time, the complexity of our approach turns out to be $O(|r| \cdot |R|^2)$ and $O(|r| \cdot |R|)$ in the worst and best cases respectively.

The enumeration strategy described above for traditional FDs relies on the ordering of attributes. Unfortunately, when working with AFDs, we must allow some tolerance on attribute cardinalities (hence, on the ordering of attributes) to accommodate possible errors in data. Consider two attributes a and b such that $|a| \gtrsim |b|$. If we were searching for FDs, we would check for $a \rightarrow b$ and not for $b \rightarrow a$ ($Z[b, a]$ lies in the lower-left part of Z and would be skipped). Conversely, when looking for AFDs, we must also consider the possibility that the higher cardinality of a is due to some errors in data; in other words, we must also check for $b \rightsquigarrow a$. In practice, this situation may occur if $|a| - \epsilon < |b| < |a|$. So, to preserve the correctness of our enumeration strategy when dealing with

AFDs, we must check both cells $Z[a, b]$ and $Z[b, a]$ whenever $abs(|a| - |b|) < \epsilon$. Obviously, as a side effect, our pruning capability will be slightly reduced since some more cells need to be checked; however, the best and worst complexity remain unchanged.

As mentioned at the beginning of this section, in this phase our goal is to detect the FDs holding between hubs related by a link l, which we actually achieve by detecting the AFDs involving the foreign keys in l. Specifically, given dv-schema $\mathcal{V} = (T, F)$, let $l \in T_L$ be a link that connects hubs $h_1, \ldots, h_n \in T_H$, which means that l includes n foreign keys, $k_1, \ldots k_n$, where k_i references hub h_i. Considering Definition 1, this already implies $l \rightarrow h_i$ for $i = 1, \ldots, n$. Additionally, we will say that $h_i \rightarrow h_j$ $(1 \leq i, j \leq n,\ i \neq j)$ if $k_i \rightsquigarrow k_j$. All the FDs determined are stored into a metadata repository, to be used at the next step for md-schema discovery and ranking. Note that, with reference to the complexity of detecting these AFDs, it is $|R| \equiv n$ and $|r| \equiv |l|$.

Example 3. In our sale example, we can realistically assume that an order is made by one customer and that a customer belongs to one class. A customer normally issues several orders, each normally including several lines. Finally, the company will reasonably have more customers than employees. So, for instance, within link L_CustOrder it must be |OrderSID| > |CustomerSID| > |EmployeeSID|. The first AFD checked is OrderSID $\rightsquigarrow$ CustomerSID, which is found to be true. Then CustomerSID $\rightsquigarrow$ EmployeeSID is checked, and we assume it does not hold. Finally, OrderSID $\rightsquigarrow$ EmployeeSID is checked, and again we assume that this does not hold in our application domain (i.e., several employees may be involved in the same order). We assume that overall, based on the data stored, two additional FDs are discovered for the sale dv-schema, namely H_Order $\rightarrow$ H_Customer and H_Customer $\rightarrow$ H_Class (a customer belongs to one class). These two FDs are shown in thick dotted lines in Fig. 1.

5.2 Md-Schema Discovery and Ranking

This process determines which elements of the source dv-schema are candidate to play the role of fact and, for each of them, creates an md-schema. Since the number of candidate facts may be large, the corresponding md-schemata are heuristically ranked before they are presented to the user.

Candidate Selection. The selection of candidates is based on two specific features of the data vault model:

– A satellite s contains a foreign key referencing the associated hub or link t, which means that each tuple of s is related to exactly one tuple of t $(s \rightarrow t)$ but several tuples of s are associated to the same tuple of t. However, since satellite are normally used to historicize attribute values, we can safely assume that, at each point in time, at most one tuple of s is valid for each tuple of t, i.e., that $t \rightarrow s$.

Algorithm 1. *MDSConstruction*($\mathcal{V}$)

Require: A dv-schema $\mathcal{V} = (T, F)$
Ensure: A set of md-schemata $\{\mathcal{M}_l\}$
1: **for all** $l \in T_L$ **do** ▷ For each potential fact l...
2: $A \leftarrow \{l\} \cup BusAttr(l)$
3: $E \leftarrow \{\langle l, a\rangle \mid a \in BusAttr(l)\}$
4: $\mathcal{M}_l \leftarrow (A, E)$ ▷ ...initialize the md-schema with fact l...
5: **for all** $h \in T_H \mid \langle l, h\rangle \in F$ **do**
6: $\mathcal{M}_l \leftarrow Explore(\mathcal{V}, \mathcal{M}_l, l, h)$ ▷ ...and build a DAG
7: **return** $\{\mathcal{M}_l\}$

– A hub h is connected to at least one link l (unless it is disconnected from all other business concepts, in which case it is most probably not a fact candidate), and $l \rightarrow h$.

It follows that, for each satellite and hub in a dv-schema, there exists a link from which that satellite or hub can be reached through at most two FDs (in case of a satellite s of a hub h, it is $l \rightarrow h \rightarrow s$). So, since the algorithm we will use to build an md-schema for each fact navigates FDs, we can restrict the set of fact candidates to the set T_L of links without loss of generality.

Md-Schema Construction. The goal of this step is to automatically build, for each candidate fact (i.e., for each link) a draft md-schema starting from the dv-schema and from the additional FDs previously discovered. To this end, all the FDs (both those explicitly modeled by the dv-schema and the additional ones discovered by accessing data) must be "navigated" starting from the candidate fact, to build a DAG of attributes that will then be ranked and enriched in the next phase to become an md-schema.

The pseudo-code for building draft md-schemata is sketched in Algorithms 1 and 2. Algorithm 1 iterates on all links in the source dv-schema. For each link l, it initializes a draft md-schema $\mathcal{M}_l$ with fact l, adds the attributes of the satellites of l (if any), and triggers procedure *Explore* to recursively build a hierarchy for each hub connected to l.

The goal of Algorithm 2 is to extend $\mathcal{M}_l$ by "exploring" hub h. First it creates a node labelled with the business key of h, k, and attaches it to the previous node g (lines 1–3). All the attributes of its satellites are then attached to k (lines 6–7). To continue exploration, the algorithm now checks if there are additional FDs from h to some other hub (lines 8–18). In particular, if there is an FD to at least one hub z through link l, before triggering recursion on z (line 18) all the satellite attributes of l must be added as children of k (lines 12–15). Repeated explorations of parts of the source dv-schema when the same hub is reached twice from different directions are avoided by marking a hub as explored when it is reached for the first time (lines 4–5).

Example 4. In our sale example, three draft md-schemata are built for facts L_LineItem, L_CustOrder, and L_CustClass (two of them are shown in Fig. 3).

To better describe the construction algorithms, we follow them step by step with reference to the first md-schema (the one of fact L_LineItem). Firstly,

Algorithm 2. *Explore*$(\mathcal{V}, \mathcal{M}_l, g, h)$

Require: A dv-schema $\mathcal{V}$, an md-schema $\mathcal{M}_l$, a node $g \in \mathcal{M}_l$, and a hub $h \in T_H$
Ensure: An (extended) md-schema $\mathcal{M}_l$

1: $k \leftarrow BusKey(h)$
2: $A \leftarrow A \cup \{k\}$ ▷ Add business key k...
3: $E \leftarrow E \cup \{\langle g, k\rangle\}$ ▷ ...and its incoming arc to $\mathcal{M}_l$
4: **if** h not explored yet **then**
5: Mark h as explored
6: $A \leftarrow A \cup BusAttr(h)$ ▷ Add satellite attributes...
7: $E \leftarrow E \cup \{\langle k, a\rangle \mid a \in BusAttr(h)\}$ ▷ ...and their arcs to $\mathcal{M}_l$
8: **for all** $l \in T_L \mid \langle l, h\rangle \in F$ **do** ▷ For each link l connected to h...
9: $Z \leftarrow \{z \in T_H \mid z \neq h \wedge \langle l, z\rangle \in F\}$ ▷ ...find other hubs connected to l
10: **if** $\exists z \in Z \mid h \rightarrow z$ **then**
11: $A \leftarrow A \cup BusAttr(l)$
12: $E \leftarrow E \cup \{\langle k, a\rangle \mid a \in BusAttr(l)\}$ ▷ Add satellite attributes of l to $\mathcal{M}_l$
13: **for all** $z \in Z \mid h \rightarrow z$ **do** ▷ Use additional FDs to trigger recursion
14: $\mathcal{M}_l \leftarrow Explore(\mathcal{V}, \mathcal{M}_l, k, z)$
15: **return** $\mathcal{M}_l$

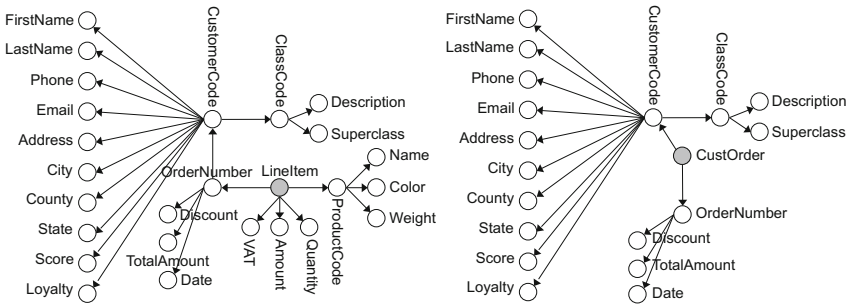

Fig. 3. Draft md-schemata of facts L_LineItem and L_CustOrder

procedure *MDSConstruction* creates the fact node (in grey) and its satellite children VAT, Amount, and Quantity. Then, procedure *Explore* is called twice for hubs H_Order and H_Product. In the first case, *Explore* starts by creating node OrderNumber (line 2), connecting it to node LineItem (line 3), and adding the two satellite children (lines 6–7). Then, since link L_CustOrder is connected to H_Order and FD H_Order → L_CustOrder holds (lines 8–12), *Explore* is called for hub H_Customer (L_CustOrder has no satellites, so lines 13-15 have no effect). When *Explore* is called for H_Customer, 10 satellite children are added, then the procedure is called again for hub H_Class. Similarly for hub H_Product.

Ranking. At the previous step, for each candidate fact l a draft md-schema $\mathcal{M}_l = (A_l, E_l)$ has been constructed. Now, the md-schemata obtained are ranked to support the user in choosing the most comprehensive ones.

The ranking of md-schemata is based on a linear combination of three heuristics that consider, for each candidate fact, (i) its cardinality, (ii) the number of potential measures, and (iii) the number of potential attributes. While heuristics (i) is extensional in nature because it is data-based, the remaining two (which are partially inspired by [12]) are intensional because they consider the dv-schema.

(i) Business events are dynamic in nature and generated with high frequency, so the tables that store them have a large number of instances. A link $l \in T_L$ is more likely to be a fact if it has high cardinality [1].
(ii) Business events are quantitatively described by several measures, i.e., numerical attributes. We quantify the probability that a link l is a fact as the number of numerical attributes that are functionally determined from l, i.e., as the number of numerical attributes in $A_l \setminus l$.
(iii) At query time, business events are selected and aggregated by users using the dimensions and their levels. We quantify the probability that a link l is a fact as the number of non-numerical attributes that are functionally determined from l, i.e., as the number of non-numerical attributes in $A_l \setminus l$.

Note that the last heuristics closely recalls the *connection topology value*, defined in [12] as the number of entities that can be (either directly or indirectly) reached within an Entity-Relationship diagram by starting from the fact and recursively navigating many-to-one relationships.

Example 5. Heuristics (ii) and (iii) for the three sales draft md-schemata return the following values for the number of numerical and non-numerical attributes: 7, 17 (L_LineItem); 1, 13 (L_CustClass); and 3, 14 (L_CustOrder). Considering that the cardinality of link L_LineItem will surely be quite higher than the one of the other two links (the cardinality of L_CustClass is at most the same of H_Customer and a customer normally issues several orders; the cardinality of L_CustOrder is at most the same of H_Order, and an order normally has several lines), we can conclude that the top ranked md-schema is the one of fact L_LineItem whatever the weights of the linear combination of the three heuristics.

5.3 Md-Schema Enrichment

The last phase starts with the user selecting one or more draft md-schemata of interest, supported by the ranking previously obtained. Some editing is normally necessary at this stage, typically to remove uninteresting attributes from the md-schema. Specific situations such as one-to-one relationships between hubs and multiple arcs entering the same node in the md-schema must be also dealt with, as discussed in [4]. Then, measures are chosen among the numerical attributes in the md-schema. Finally, all the direct children of the fact that have not been chosen as measures are labelled as dimensions, which completely defines the output md-schema.

One further way to enrich the md-schema by making its hierarchies more faithful to the application domain is to search for FDs hidden in satellites. In a data vault, the grouping of attributes in satellites is generally oriented more to cheap maintainability and querying than to normalization. For instance, in our sale example, satellites S_CustAddress and S_Employee contain attributes City, County, and State that are obviously related to one another, so the following FDs hold: City $\rightarrow$ County and County $\rightarrow$ State. While in this simple case it will probably be easy for the user to detect these FDs and manually add them to

the md-schema as a part of editing, in other cases the user may be unsure of whether an FD holds or not, so automating FD detection is highly desirable. How to cope with this issue is the subject of the remainder of this section.

When dealing with satellites, we must keep in mind that data vaults are natively oriented to storing time-variant data, so we can expect that a single tuple of a hub (or link) is related to several tuples in a connected satellite, one for each version of data. As a consequence, if we used traditional FD (or even AFD) discovery techniques on the S_CustAddress satellite for instance, we might not find the FD City → County in case a city has been moved to a different county at some time. The most natural way to formalize this problem is by using *temporal FDs* [9]. Intuitively, in its simplest form, a temporal FD $a \xrightarrow{T} b$ is an FD that is valid within a time-variant relation at any time slice. In our example, though City → County may be not true overall, it must be true at any time slice, so City $\xrightarrow{T}$ County. If we also consider the possibility that a temporal FD holds on *most* tuples of a satellite, we have *approximate temporal FDs* (ATFDs) [2], i.e., FDs that are valid for specific time periods and possibly subject to errors.

In [2], the detection of ATFDs is achieved through some preprocessing that turns them into AFDs, that can then be discovered using Tane [7]; this preprocessing is made by temporally grouping either on sliding windows or on temporal granules. The type of temporal evolution that is relevant to the Starry Vault approach is captured by grouping on temporal granules, i.e., by partitioning the values in the domain of the time attribute into indivisible groups called *granules*. Examples of possible granularities are hours, days, months, etc. To understand how this preprocessing works, consider a table r with schema $R = v \cup W$, where v and W are respectively a time attribute and a set of other attributes. A new relation is created from r by adding a granule attribute g whose domain is the set of granules included in the time-span described by the instances of r. Intuitively, for each tuple in r, the value of v is converted into its corresponding granule identifier. The new relation obtained is then processed with Tane to discover AFDs of type $g \cup X \rightsquigarrow Y$, with $X, Y \subseteq W$.

To apply this technique to a satellite s, we consider its timestamp and its business attributes $BusAttr(s)$, thus neglecting its foreign key. After the the granule attribute g has been addded, the ATFDs can be computed using the following variation of the enumeration strategy proposed in Sect. 5.1:

- Instead of searching for AFDs of the form $a \rightsquigarrow b$, we consider all AFDs of the form $ga \rightsquigarrow gb$ (i.e., due to the decomposition rule, $ga \rightsquigarrow b$), where $a, b \in BusAttr(s)$. This means that the ordering for rows and columns in matrix Z will be defined by the cardinality of ga rather than by that of a.
- The pruning rule seen in Sect. 5.1 would avoid checking all AFDs $b \rightsquigarrow a$ with $|a| > |b| + \epsilon$. Conversely, in this case a check can be avoided if $|ga| > |gb| + \epsilon$.

It is easy to see that the size of matrix Z is still $|R|^2 \equiv |BusAttr(s)|^2$ since we are just adding the granule attribute g to both the left- and right-hand sides of the AFDs. As to the correctness of the pruning rule, we remark that the error $e(ga \rightsquigarrow b)$ is defined as the minimum number of distinct values of gab that must

Table 1. Sample data for the S_CustAddress satellite

CustomerSID	Timestamp	Address	City	County	State	Granule
1	1-1-2015	Gandalf Street	Minas Tirith	Gondor	Middle-Earth	January 2015
1	1-6-2015	Gandalf Street	Minas Tirith	Rohan	Middle-Earth	June 2015
2	1-3-2015	Frodo Road	Minas Tirith	Gondor	Middle-Earth	March 2015
2	1-6-2015	Frodo Road	Minas Tirith	Rohan	Middle-Earth	June 2015

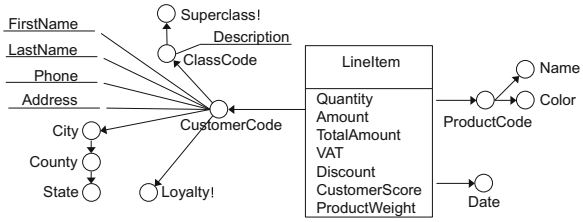

Fig. 4. The enriched md-schemata of fact S_LineItem (descriptive attributes, non usable for aggregation, are underlined)

be removed to enforce $ga \rightarrow b$; therefore, an error ϵ can at most impact on the cardinality of b for an amount equal to ϵ itself.

Example 6. Consider the sample data for the S_CustAddress satellite in Table 1, showing that on June 1 the city of Minas Tirith has moved from the Gondor county to that of Rohan. If we considered traditional FDs or even AFDs, we would probably conclude that one city can belong to different counties (i.e., that City $\nrightarrow$ County). Let us consider ATFDs instead, choosing for instance a month granularity. The table created after preprocessing has the new column Granule, and it is easy to verify that Granule City $\rightarrow$ County, so City $\xrightarrow{T}$ County. The final md-schema obtained from the draft md-schema of fact L_LineItem (Fig. 3, top) is depicted in Fig. 4 using the DFM notation [4]. Attribute OrderNumber has been deleted and all numerical attributes have been chosen as measures; besides, the missing FDs between City, County, and State have been added.

6 Conclusions

In this paper we have described the Starry Vault approach for detecting a multidimensional schema out of a source data vault. Both schema-based and data-based FDs are used to this end, with a small intervention by the user. In particular we have shown how to use extensional techniques for discovering hidden FDs, with some tolerance to errors in data and taking into account the temporal aspects related to historicization, to automatically deliver the md-schemata that better fit the business domain. To this end we have proposed an original exploration strategy that allows to significantly reduce the complexity of the Tane

algorithm when applied to simple ATFDs. To the best of our knowledge, ours is the first approach that adopts advanced types of FDs to infer md-schemata.

Automatic derivation of md-schemata is a widely explored topic in the DW literature; nonetheless we believe that it is worth reconsidering it in the era of big data and data science, in which the need for on-the-fly analyses creates a strong requirement for a smarter design process. Based on these considerations, our future work on this topic will be mainly focused on investigating ad hoc techniques to support the data scientist in discovering a multidimensional structure even in situations in which the source data are poorly-structured or schemaless, as is the case for document databases.

References

1. Carmè, A., Mazón, J.-N., Rizzi, S.: A model-driven heuristic approach for detecting multidimensional facts in relational data sources. In: Bach Pedersen, T., Mohania, M.K., Tjoa, A.M. (eds.) DAWAK 2010. LNCS, vol. 6263, pp. 13–24. Springer, Heidelberg (2010)
2. Combi, C., Parise, P., Sala, P., Pozzi, G.: Mining approximate temporal functional dependencies based on pure temporal grouping. In: Proceedings of ICDM Workshops, pp. 258–265, Dallas, USA (2013)
3. Golfarelli, M., Maio, D., Rizzi, S.: Conceptual design of data warehouses from E/R schemes. In: Proceedings of HICSS, pp. 334–343, Kohala Coast, HI (1998)
4. Golfarelli, M., Rizzi, S.: Data Warehouse Design: Modern Principles and Methodologies. McGraw-Hill, New York (2009)
5. Golfarelli, M., Rizzi, S., Vrdoljak, B.: Data warehouse design from XML sources. In: Proceedings of DOLAP, pp. 40–47, Atlanta, Georgia (2001)
6. Hughes, R.: Agile Data Warehousing for the Enterprise. Elsevier Science, Amsterdam (2015)
7. Huhtala, Y., Kärkkäinen, J., Porkka, P., Toivonen, H.: TANE: an efficient algorithm for discovering functional and approximate dependencies. Comput. J. **42**(2), 100–111 (1999)
8. Hultgren, H.: Data vault modeling guide (2012). http://hanshultgren.files.wordpress.com
9. Jensen, C.S., Snodgrass, R.T., Soo, M.D.: Extending existing dependency theory to temporal databases. IEEE Trans. Knowl. Data Eng. **8**(4), 563–582 (1996)
10. Jensen, M.R., Holmgren, T., Pedersen, T.B.: Discovering multidimensional structure in relational data. In: Kambayashi, Y., Mohania, M., Wöß, W. (eds.) DaWaK 2004. LNCS, vol. 3181, pp. 138–148. Springer, Heidelberg (2004)
11. Jovanovic, V., Bojicic, I.: Conceptual data vault model. In: Proceedings of SAIS, vol. 23, pp. 1–6, Atlanta, Georgia (2012)
12. Kim, J., et al.: SAMSTARplus: an automatic tool for generating multi-dimensional schemas from an entity-relationship diagram. Revista de Informática Teórica e Aplicada **16**(2), 79–82 (2009)
13. Krneta, D., Jovanovic, V., Marjanovic, Z.: A direct approach to physical data vault design. Comput. Sci. Inf. Syst. **11**(2), 569–599 (2014)
14. Linstedt, D.: DV modeling specification v1.09 (2013). http://danlinstedt.com
15. Phipps, C., Davis, K.C.: Automating data warehouse conceptual schema design and evaluation. In: Proceedings of DMDW, pp. 23–32, Toronto, Canada (2002)

16. QOSQO: QUIPU 1.1 Whitepaper (2016). www.datawarehousemanagement.org
17. Romero, O., Abelló, A.: A framework for multidimensional design of data warehouses from ontologies. Data Knowl. Eng. **69**(11), 1138–1157 (2010)
18. Winter, R., Strauch, B.: A method for demand-driven information requirements analysis in data warehousing projects. In: Proceedings of HICSS, p. 231, Big Island (2003)