Ontology-Based Multidimensional Contexts with Applications to Quality Data Specification and Extraction

Mostafa Milani and **Leopoldo Bertossi**
Carleton University, School of Computer Science
Ottawa, Canada

Abstract. Data quality assessment and data cleaning are context dependent activities. Starting from this observation, in previous work a context model for the assessment of the quality of a database was proposed. A context takes the form of a possibly virtual database or a data integration system into which the database under assessment is mapped, for additional analysis, processing, and quality data extraction. In this work, we extend contexts with dimensions, and by doing so, multidimensional data quality assessment becomes possible. At the core of multidimensional contexts we find ontologies written as Datalog$^\pm$ programs with provably good properties in terms of query answering. We use this language to represent dimension hierarchies, dimensional constraints, dimensional rules, and specifying quality data. Query answering relies on- and triggers dimensional navigation, and becomes an important tool for the extraction of quality data.

1 Introduction

Data quality assessment and data cleaning are context-dependent activities. More precisely, the quality of data has to be assessed with some form of contextual knowledge, in particular, about the *production and the use* of data, among other possible dimensions of data quality. Data quality refers to the degree in which data fits or fulfills a form of usage [3, 22]. As expected, context-based data quality assessment requires a formal model of context. Accordingly, we propose a model of context that addresses quality concerns that are related to the production and use of data.

Here we follow and extend the approach in [4] that provides a model of context for data quality assessment. In that work, the assessment of a database D is performed by *putting D in context*, more precisely, by mapping it into a context $\mathcal{C}$ (Fig. 1, left), which is represented as another database, or as a database schema with partial information, or, more generally, as a virtual data integration system [24]. The latter may have some materialized data and access to external data sources.

The quality of data in D is determined through additional processing, material or virtual, of the data within the context. These contextual data may be imported from D or may be already available at the context. The context may also contain application-dependent knowledge associated to data quality, in the form of rules or semantic constraints. Data processing in the context leads to possible several quality versions of D, forming a class $\mathcal{D}^q$ of intended, clean versions of D (Fig. 1, right). The quality of D is measured in terms of how much D departs from (its quality versions in) $\mathcal{D}^q$: $dist(D, \mathcal{D}^q)$. Of course, different distance measures may be used for this purpose [4].

In some cases, we may want to assess the quality of answers to a query $\mathcal{Q}$ posed to instance D or to obtain "quality answers" from D. This can be done appealing to the class $\mathcal{D}^q$ of intended clean versions of D. For assessment, the set of query answers to

$\mathcal{Q}$ from D can be compared with the *certain answers* for $\mathcal{Q}$, i.e. the intersection of the sets of answers to $\mathcal{Q}$ from each of the instances in $\mathcal{D}^q$ [21]. The certain answers become what we could call the *clean answers* to $\mathcal{Q}$ from D [4]. So, if we want the clean answers to $\mathcal{Q}$ from D, instead of computing the answers from D as usual, we compute the clean answers (cf. bottom of Fig. 1).

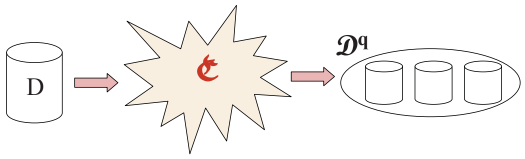


Fig. 1. Clean instances and query answers

When computing clean query answers, instead of computing, materializing and querying all the instances in class $\mathcal{D}$, a form of *query rewriting* can be attempted: a new query $\mathcal{Q}^q$ is posed to D to obtain the clean answers for $\mathcal{Q}$. Some cases of rewriting were investigated in [4]. In this work we continue adopting this approach to data quality assessment and clean query answering. However, as we will see, the contexts we consider in this work are more complex than those considered in [4], and for good reasons.

Actually, an important contextual element was not considered in [4]: *dimensions*. They were *not* considered as contextual elements for data quality analysis, but in practice, dimensions are naturally associated to contexts. Here, in order to capture general dimensional aspects of data for inclusion in contexts, we take advantage of- and start from the Hurtado-Mendelzon (HM) multidimensional data model [20], whose inception was mainly motivated by data warehouses (DWH) and OLAP applications.

We extend the HM model by adding *categorical relations* associated to categories, at different levels of the dimension hierarchies, possibly to more than one dimension (think of generalized fact tables as found in data warehouses). It also include *dimensional constraints* and *dimensional rules,* which could be treated both as *dimensional integrity constraints* on categorical relations that involve values from dimension categories. However, dimensional constraints are intended to be used as *denial constraints* that forbid certain combinations of values, whereas the dimensional rules are intended to be used for data completion, to generate data through their enforcement via *dimensional navigation*.

In this work we propose an ontological representation in Datalog$^\pm$ [8, 9] of the extended HM model, and also mechanisms for data quality assessment based on query answering from the ontology via dimensional navigation. As already suggested, the idea is that a query to the ontology triggers dimensional navigation and the creation of missing data, in possible upward and downward directions, and on multiple dimensions. Datalog$^\pm$ supports data generation through the ontological rules. This is particularly useful, and also much in line with the way we understand and use contexts in everyday life: *Contexts allows us to extend or expand information that, otherwise, without this extension, would be impossible or difficult to understand or make sense of*. Furthermore, this ontological approach captures well our general philosophy according to which, *contexts should be represented as formal theories into which other objects, like database instances, are mapped*, for contextual analysis, assessment, interpretation, and additional processing [4].

Datalog$\pm$ is an extension of classical Datalog, mainly through the use of existentially quantified variables (aka. value invention) in rule heads. It has been successfully applied to the logical representations of data models and ontologies [11, 13]. Actually, a *multidimensional context* -corresponding to the formalization of the extension of HM- becomes a Datalog$\pm$ ontology, $\mathcal{M}$, that belongs to an interesting syntactic class of programs, for which some results are known. This allows us to give a semantics to our ontologies, and apply some established and new algorithms for query answering.

More precisely, the core multidimensional ontology $\mathcal{M}$ is a weakly-sticky Datalog$^\pm$ program [12], for which (conjunctive) query answering has polynomial-time data complexity. In our case, weak-stickiness is due to the -as we argue, natural- assumptions that: (a) dimension navigation (as captured by data generation) happens through rules with body joins on *categorical attributes* (i.e. in categorical relations), whose values come from dimension categories; and (b) there is no value invention for categorical attributes. (We also discuss cases where these assumptions do not hold.)

Multidimensional ontologies are used to support quality data specification and extraction.[1] More precisely, and continuing with the above idea on this use of contexts, it amounts to: (a) defining application-dependent *quality predicates* (they can be seen as views capturing data quality concerns), (b) using them to define the *quality versions* of the original predicates (relations) in the database D under quality assessment, and (c) retrieving quality data by querying the (possibly virtual extensions of the) latter predicates [4]. These predicate definitions may be based on *data quality guidelines* that are captured as rules or semantic constraints, both of which may refer to categorical attributes of predicates in $\mathcal{M}$, without being part of $\mathcal{M}$. Rather, this "quality part" of the context comes on top of $\mathcal{M}$. We establish that under reasonable conditions on these extra definitions, the resulting extension of $\mathcal{M}$ still retains the tractability of query answering (even when weak-stickiness may be compromised).

About related work, in [6] dimensions become the basis for *building* contexts, or more precisely database instances that are tailored according to certain dimensional elements. This is done through a process of selection of relevant dimensional elements: the dimension leaves a footprint on the data. As a result, the constructed database is implicitly dimensional, and the dimensions as such may be lost as first-class objects in the generated context.

In [26, 27] the authors consider the generation of data at different levels of a category hierarchy, and at query answering time. This involves hierarchy navigation and an extension of relational algebra that computes data by appealing to data at other levels of the hierarchy. Actually, in our work we show how this process can be captured via our Datalog$^\pm$ MD ontologies.

DWHs have been represented in expressive description logics (DL) [16, 17]. Preliminary research on extensions in DL of the HM model, also for data quality purposes, can be found in [23].

Summarizing, in this work we make the following contributions:[2]

1. We extend HM data model and represent the extension as a Datalog$^\pm$ ontology that contains: (a) categorical relations, (b) tuple-generating-dependencies, *TGDs* (a rule

[1] In this work we do not explicitly address the problem of assessing the quality of the original data through a numerical comparison with the quality data [4].

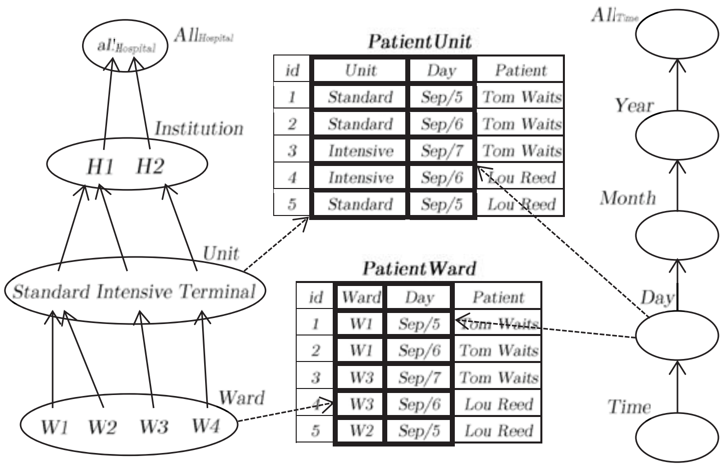
[2] This work considerably extends [28], which contains basically the material of Section 2 here.

incarnation of referential constraints), to connect the original data to categorical relations, and the latter to dimensions; and (c) dimensional constraints.

2. We establish that the multidimensional ontology is a *weakly-sticky* Datalog$^\pm$ program [12]. As a consequence, query answering can be done in polynomial time.
3. We analyze the effect of dimensional constraints on query answering, specifically the *separability condition* [12] between *TGDs* and constraints that are equality-genera-ting-dependencies, *EGDs*. We show that, by restricting variables in equalities to appear categorical attributes, separability holds.
4. We propose a general approach for contextual data quality specification and extraction that is based on MD ontologies, emphasizing the dimensional navigation process that is triggered by queries about quality data. We illustrate the application of this approach by means of an extended example.

2 An Extended, Motivating Example

This section illustrates the intuition behind categorical relations, dimensional rules and constraints, and how they are used for data quality purposes. We assume, according to the HM model (cf. Section 3), that a dimension consists of a finite set of categories related to each other by a partial order.



PatientUnit

id	Unit	Day	Patient
1	Standard	Sep/5	Tom Waits
2	Standard	Sep/6	Tom Waits
3	Intensive	Sep/7	Tom Waits
4	Intensive	Sep/6	Lou Reed
5	Standard	Sep/5	Lou Reed

PatientWard

id	Ward	Day	Patient
1	W1	Sep/5	Tom Waits
2	W1	Sep/6	Tom Waits
3	W3	Sep/7	Tom Waits
4	W3	Sep/6	Lou Reed
5	W2	Sep/5	Lou Reed

Fig. 2. An extended multidimensional model

Example 1. The relational table *Measurements* (Table 1) shows body temperatures of patients in an institution. A doctor wants to know *"The body temperatures of Tom Waits for September 5 taken around noon with a thermometer of brand B1"* (as he expected). Possible a nurse, unaware of this requirement, used a thermometer of brand *B2*, storing the data in *Measurements*. In this case, not all the measurements in the table are up to the expected quality. However, table *Measurements* alone does not discriminate between intended values (those taken with brand *B1*) and the others.

For assessing the quality of the data in *Measurements* according to the doctor's quality requirement, extra contextual information about the thermometers in use may help. In this case, the table *PatientWard*, linked to the *Ward* category (Fig. 2, middle, bottom). This *categorical relation* stores patient names for each ward of the institution.

Furthermore, the institution has a *guideline* prescribing that: *"Temperature measurement for patients in a standard care unit have to be taken with thermometers of Brand B1"*. It can be used for data quality assessment when combined with categorical table *PatientUnit* (Fig. 2, middle, top), which is linked to the *Unit* category, and whose data are (at least partially) generated from *PatientWard* by upward-navigation through dimension Hospital (Fig. 2, left), from category *Ward* to category *Unit*.

Table 1. *Measurements*

	Time	Patient	Value
1	Sep/5-12:10	Tom Waits	38.2
2	Sep/6-11:50	Tom Waits	37.1
3	Sep/7-12:15	Tom Waits	37.7
4	Sep/9-12:00	Tom Waits	37.0
5	Sep/6-11:05	Lou Reed	37.5
6	Sep/5-12:05	Lou Reed	38.0

According to the guideline, it is now possible to conclude that, on days when Tom Waits was in the standard care unit, his temperature values were taken with the expected thermometer: for patients in wards *W1* or *W2* a thermometer of brand *B1* was used. These "clean data" -in relation to the doctor's expectations- appear in relation $Measurements^q$ (Table 2).

Table 2. $Measurements^q$

	Time	Patient	Value
1	Sep/5-12:10	Tom Waits	38.2
2	Sep/6-11:50	Tom Waits	37.1

Elaborating on this example, there could be a *dimensional constraint*: *"No patient in intensive care unit at any time after August/2005"*. As stated, this constraint could be represented as a "static" constraint on the categorical relation *PatientUnit*. However, it could also be represented as one on the data generation process via upward-navigation from *PatientWard* to *PatientUnit*, preventing the use of the third tuple in table *PatientWard*. As such, this becomes a *navigational constraint* that also involves dimensions Hospital and Time (Fig. 2, right). A third alternative is handling the constraint as a "static" constraint on the join of *PatientWard* and *PatientUnit* via the patient name (*Tom Waits* could not be both in ward *W3* and intensive care on some dates). Our approach will allow to handle the constraint in any of these three forms. ■

Categorical relations may be incomplete, and new data can be generated for them, which will be enabled through rules (*tgds*) of a Datalog$\pm$ dimensional ontology. The previous example shows data generation via upward navigation. Our next example shows that *downward navigation* may also be useful. Our approach to multidimensional contexts will support both.

Example 2. (ex. 1 cont.) Consider two additional categorical relations, *WorkingSchedules* (Table 3) and *Shifts* (Table 4), linked to categories *Unit* and *Ward*, resp. They store schedules of nurses in units and shifts of nurses in wards, resp. A query to *Shifts* asks for dates when *Mark* was working in ward *W2*, which has no answer with the data in Table 4. A new guideline states: *"If a nurse works in a unit on a specific day, he/she has shifts in every ward of that unit on the same day"*. It can be captured as a dimensional rule connecting *WorkingSchedules* to *Shifts* via the dimension hierarchy. Downward data generation using this rule, tuple 5 in Table 3, and the dimensional connection of *Standard* to *W1*, *W2*, makes *Mark* have shifts in both *W1* and *W2* on *Sep/9*. ■

Table 3. *WorkingSchedules*

	Unit	Day	Nurse	Type
1	Intensive	Sep/5	Cathy	cert.
2	Standard	Sep/5	Helen	cert.
3	Standard	Sep/6	Helen	cert.
4	Terminal	Sep/5	Susan	non-c.
5	Standard	Sep/9	Mark	non-c.

Table 4. *Shifts*

	Ward	Day	Nurse	Shift
1	W4	Sep/5	Cathy	night
2	W1	Sep/6	Helen	morning
3	W4	Sep/5	Susan	evening

3 Preliminaries

We first briefly review previous work in [4] on context-based data quality assessment. The starting point is that *data quality is context dependent*. A context provides *knowledge about the way data are interrelated, produced and used*, which allows us to make sense of the data. In our view, both the database under quality assessment and the context can be formalized as logical theories. The former is then *put in context* by mapping it into the latter, though logical mappings and possibly shared predicates.

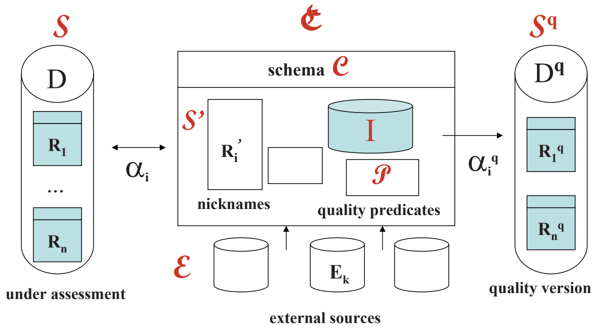


Fig. 3. A context for data quality assessment

In Fig. 3, D is a relational database (with schema $\mathcal{S}$) under quality assessment. It can be represented as a logical theory [31]. The context, $\mathfrak{C}$ in the middle, resembles a virtual data integration system, which can also be represented as a logical theory [24]. The context has a relational schema (or signature), $\mathcal{C}$, in particular predicates with possibly partial extensions (incomplete relations). The mappings in between are of the kind used in data integration or data exchange [19], that can be expressed as logical formulas. In this paper, we are not concerned with how such a context is created [4].

A subschema of $\mathcal{C}$ may have an instance I, but $\mathcal{C}$ have nicknames (copies) R' for predicates R in $\mathcal{S}$. Nicknames are used to to map (via the α_i) the data in D into $\mathfrak{C}$, for further logical processing. So, schema $\mathcal{C}$ can be seen as an expansion of $\mathcal{S}$ through a subschema $\mathcal{S}'$. Some predicates in $\mathcal{C}$ are meant to be *quality predicates* (in $\mathcal{P}$), which are used to specify single quality requirements. There may be semantic constraints on schema $\mathcal{C}$, and also access (mappings) to external data sources, in $\mathcal{E}$, that could be used for data assessment or cleaning.

A clean version of D, obtained through the mapping into- and processing within context $\mathfrak{C}$, is a possibly virtual instance D^q (or a collection thereof, as suggested in Fig. 1), for schema $\mathcal{S}^q$ (a "quality" copy of schema $\mathcal{S}$). The extension of every predicate in it, say R^q, is the "quality version" of relation R in D, and is defined as a view (via the α_i^q) in terms of the nickname predicates in $\mathcal{S}'$, those in $\mathcal{P}$, and other contextual predicates. The quality of (the data in) instance D can be measured by comparing D with the instance D^q or the set, $\mathcal{D}^q$, of them. This latter set can also be used to define and possibly compute the *quality answers* to queries originally posed to D, as the *certain* *answers* wrt. $\mathcal{D}^q$. See [4] for more details, and different cases that may occur. In any case, the main idea is that quality data can be extracted from D by querying the possibly virtual class $\mathcal{D}^q$.

In this paper, we extend the approach to data quality specification and extraction we just described, by adding dimensions to contexts, for multidimensional data quality specification and extraction. In this case, the context contains a generic multidimensional ontology, the shaded $\mathcal{M}$ in Fig. 4, aka. "core ontology" (and described in Section 4). This ontology can be extended, within the context, with additional rules and constraints that depend on specific data quality concerns (cf. Section 6).

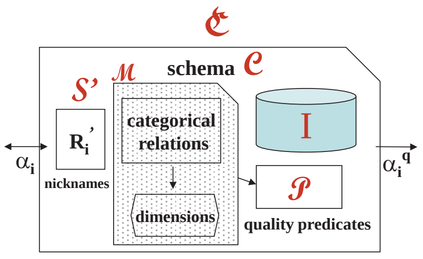


Fig. 4. A multidimensional context

According to the Hurtado-Mendelzon (HM) multidimensional data model [20], a *dimension schema*, $\mathcal{S} = \langle \mathcal{K}, \nearrow \rangle$, is a directed acyclic graph (a lattice), with $\mathcal{K}$ a set of categories (represented as unary predicates), and $\nearrow$ the *parent-child relation* between categories. $\nearrow^*$ denotes the transitive and reflexive closure of $\nearrow$, and is a partial order with a *top category*, *All*, which is reachable from every other category. There is a unique *base category*, which does not have children. A *dimension instance* for schema $\mathcal{S}$ is a tuple $\mathcal{D} = \langle \mathcal{N}, <, \sigma \rangle$, with $\mathcal{N}$ a set of *elements*, $<$ is a *parent-child relation* between elements, and $\sigma : \mathcal{N} \rightarrow \mathcal{K}$, the *membership function*, is total and injective. A dimension instance is shown in Fig. 2, left. The partial order $<$ parallels (is consistent with) $\nearrow$: $a < b$ implies $\sigma(a) \nearrow \sigma(b)$. $\sigma(e) = k$ is also denoted as $e \in k$ or $k(e)$ (holds). $<^*$ is the transitive and reflexive closure of $<$, and is used to define the *roll-up* relations for any pair of categories k and k': $L_k^{k'}(\mathcal{D}) = \{(e, e') \mid e \in k,\ e' \in k' \text{ and } e <^* e'\}$.

Datalog$^\pm$ [8, 9] is a family of rule languages that properly extends plain Datalog: (a) rules (aka. *tgds*) may have existential quantifiers in the heads; (b) *equality-generating dependencies* (*egds*), i.e. rules with only equality in the head; and (c) *negative constraints* (NCs), that are rules with $\perp$, a false propositional atom, in the heads, indicating that the rule body cannot be true.

Example 3. This Datalog$^\pm$ program shows a *tgd*, an *egd*, and an *NC*, in this order: $\exists x Assist(d, x) \leftarrow Doctor(d)$; $x = x' \leftarrow Assist(d, x), Assist(d, x')$; $\perp \leftarrow Specialist(d, x, n), Nurse(d, n)$. ■

Datalog$^\pm$ has been used to represent ontological knowledge and conceptual data models [11, 13]; and for *ontology-based data access* [15, 18]. The underlying extensional, relational database (the facts) $\mathcal{I}$ for a program may be incomplete, and the *chase* is the standard procedure for completing the database, through the enforcement of the program rules. When a *tgd* is applied, new atoms are created, possibly including fresh nulls (for the existential variables), and the whole run of the chase may be non-terminating, leading to an infinite complete database. The enforcement of an *egd* equates nulls with nulls or nulls with constants or fails. For a set Σ of *tgds* and *egds*, $chase(\mathcal{I}, \Sigma)$ denotes the possibly infinite instance resulting from the non-failing chase of Σ on $\mathcal{I}$.

Even with an infinite $chase(\mathcal{I}, \Sigma)$ it is possible that *conjunctive query answering* (QA) is decidable (or computable). The $^-$ in Datalog$^\pm$ stands for syntactic restrictions on the interaction of *tgds* in Σ that ensure decidability of QA, and, in some cases, also tractability (in data). So, Datalog$^\pm$ is family of languages, with different degrees of expressivity and computational properties. Some of them are: *linear*, *guarded*, *weakly-guarded*, *sticky*, and *weakly-sticky* Datalog$^\pm$ [8, 9, 10, 11, 12]. In this work (cf. [30, appendix A]), we are particularly interested in *weakly-sticky* (WS) Datalog$^\pm$ [12], which extends *sticky* Datalog$^\pm$ [10].

4 Extending the HM Model with Datalog$^\pm$

We extend the HM model introducing *categorical relations*, each of them having a relational schema with a name, and attributes, some of which are *categorical* and the other, *non-categorical*. The former take values that are members of a dimension category. The latter take values from an arbitrary domain. Categorical relations have to be logically connected to dimensions. For this we use a Datalog± ontology $\mathcal{M}$, which has a relational schema $\mathcal{S}_\mathcal{M}$, an instance $\mathcal{D}_\mathcal{M}$, and a set $\Sigma_\mathcal{M}$ of dimensional rules, and a set $\kappa_\mathcal{M}$ of constraints. Here, $\mathcal{S}_\mathcal{M} = \mathcal{K} \cup \mathcal{O} \cup \mathcal{R}$, with $\mathcal{K}$ a set of unary *category predicates*, $\mathcal{O}$ a set of *parent-child predicates*, capturing $<$-relationships for pairs of adjacent categories, and $\mathcal{R}$ a set of *categorical predicates*, say $R(C_1, \ldots; N_1, \ldots)$, where, to highlight, categorical and non-categorical attributes (C_is vs. N_js) are separated by ";".

Example 4. Categorical relation $PatientWard(Ward,Day;Patient)$ in Fig. 2 has categorical attributes *Ward* and *Day*, connected to the Hospital and Time dimensions, resp. *Patient* is non-categorical. $Ward(\cdot)$, $Unit(\cdot) \in \mathcal{K}$; $\mathcal{O}$ contains, e.g. a binary predicate connecting *Ward* to *Unit*; and $\mathcal{R}$ contains, e.g. *PatientWard*. ■

The (extensional) data, $\mathcal{D}_\mathcal{M}$, associated to the ontology $\mathcal{M}$'s schema are the complete extensions for categories in $\mathcal{K}$ and predicates in $\mathcal{O}$ that come from the dimension instances. The categorical relations (with predicates in $\mathcal{R}$) may contain partial data, i.e. they may have incomplete. They can belong to instance I in Fig. 4. Dimensional rules in $\Sigma_\mathcal{M}$ are those in (c) below; and constraints in $\kappa_\mathcal{M}$, those in (a) and (b).

(a) *Referential constraints* between categorical attributes and categories as negative constraint:[3] ($R \in \mathcal{R}$, $K \in \mathcal{K}$; $\bar{e}, \bar{a}$ are categorical, non-categorical, resp.; $e \in \bar{e}$)

$$\bot \leftarrow R(\bar{e};\bar{a}), \neg K(e). \tag{1}$$

Notice that K, to which negation is applied, is a closed, extensional predicate.

(b) Additional *dimensional constraints*, as *egds* or *NCs*: ($R_i \in \mathcal{R}$, $D_j \in \mathcal{O}$, and x, x' stand both for either categorical or non-categorical attributes in the body of (2))

$$x = x' \leftarrow R_1(\bar{e}_1;\bar{a}_1), ..., R_n(\bar{e}_n;\bar{a}_n), D_1(e_1, e_1'), ..., D_m(e_m, e_m'). \tag{2}$$

$$\bot \leftarrow R_1(\bar{e}_1;\bar{a}_1), ..., R_n(\bar{e}_n;\bar{a}_n), D_1(e_1, e_1'), ..., D_m(e_m, e_m'). \tag{3}$$

(c) *Dimensional rules* as Datalog$^\pm$ *tgds*:

$$\exists \bar{a}_z\, R_k(\bar{e}_k;\bar{a}_k) \leftarrow R_1(\bar{e}_1;\bar{a}_1), ..., R_n(\bar{e}_n;\bar{a}_n), D_1(e_1, e_1'), ..., D_m(e_m, e_m'). \tag{4}$$

Here, $\bar{a}_z \subseteq \bar{a}_k$, $\bar{e}_k \subseteq \bar{e}_1 \cup ... \cup \bar{e}_n \cup \{e_1, ..., e_m, e_1', ..., e_m'\}$, $\bar{a}_k \smallsetminus \bar{a}_z \subseteq \bar{a}_1 \cup ... \cup \bar{a}_n$; and body joins are only between categorical attributes (in the categorical relations $R_i(\bar{e}_i;\bar{a}_i)$), and attributes in parent-child predicates $D_j(e_j, e_j')$. Value invention is only on non-categorical attributes (we will consider relaxing this later on).

[3] An alternative and more problematic approach, may use *tgds* between categorical attributes and categories, making it possible to generate elements in categories or categorical attributes.

Some of the lists in the bodies of (2)-(4) may be empty, i.e. $n = 0$ or $m = 0$. This allows us to represent, in addition to properly "navigational" constraints, also classical constraints on categorical relations, e.g. keys or FDs.

Example 5. (ex. 1 and 4 cont.) In relation *PatientUnit*, the categorical attribute *Unit* takes values from the *Unit* category. We use a constraint of the form (1), namely: $\bot \leftarrow \mathit{PatientUnit}(u, d; p), \neg \mathit{Unit}(u)$. The constraint *"No patient in intensive care unit after August 2005"* becomes a dimensional (navigation) constraint of the form (3):

$$\bot \leftarrow [\mathit{PatientWard}(w, d; p),\, \mathit{UnitWard}(\texttt{Intensive}, w), \quad \mathit{MonthDay}(\texttt{August/2005}, d)]. \tag{5}$$

Alternatively, we could apply a constraint directly on $\mathit{PatientUnit}$, without explicit navigation in the Hospital dimension, but we still need mention navigation in the Time dimension: $\bot \leftarrow \mathit{PatientUnit}(\texttt{Intensive}, d; p), \mathit{MonthDay}(\texttt{August/2005}, d)$.

An *egd* of the form (2) says that *"All thermometers in a unit are of the same type"*:

$$t = t' \leftarrow \mathit{Therm}(w, t; n),\, \mathit{Therm}(w', t'; n'), \mathit{UnitWard}(u, w), \mathit{UnitWard}(u, w') \tag{6}$$

with $\mathit{Therm}(\mathit{Ward}, \mathit{Thertype}; \mathit{Nurse})$ a categorical relation, and *Ward*, *Thertype* categorical attributes (the latter for an Instrument dimension). This *egd* illustrates the flexibility of our approach. Even without having a categorical relation at the *Unit*, we could still impose a condition at that level.[4]

The following *tgds* generate data from *PatientWard* to *PatientUnit*, and from *WorkingSchedules* to *Shifts*, resp. They are of the form (4).

$$\mathit{PatientUnit}(u, d; p) \leftarrow \mathit{PatientWard}(w, d; p),\, \mathit{UnitWard}(u, w). \tag{7}$$

$$\exists z\ \mathit{Shifts}(w, d; n, z) \leftarrow \mathit{WorkingSchedules}(u, d; n, t),\, \mathit{UnitWard}(u, w). \tag{8}$$

The existential variable in (8) makes up for the missing, non-categorical attribute in the "parent" relation *WorkingSchedules*. This is not needed in (7). ■

Remark 1. A general *tgd* of the form (4) enables *upward*- or *downward-navigation*, depending on the body joins. The direction is determined by the dimension levels of categorical attributes in the joins. For simplicity, assume that there is a single $D_j \in \mathcal{O}$ in the body (as in (7) and (8)). If the join is between $R_i(\bar{e}_i; \bar{a}_i)$ and $D_j(e_j, e'_j)$ then: (a) (one-step) upward navigation is enabled, from e'_j to e_j, when $e'_j \in \bar{e}_i$ (i.e. e'_j appears in $R_i(\bar{e}_i; \bar{a}_i)$) and $e_j \in \bar{e}_k$, i.e in the head). (b) (one-step) downward navigation is enabled, from e_j to e'_j, when e_j occurs in R_i and e'_j occurs in R_k. Several occurrences of parent-child predicates in a body capture multi-step navigation. ■

Example 6. (ex. 5 cont.) Rule (8) captures downward-navigation; and this is a general behavior with *tgds* of the form (4), when drilling-down via (8), from a tuple, say $\mathit{WorkingSchedules}(u, d; n, t)$ via the category member u (for Unit), *for each* child w of u in the *Ward* category, a tuple for *Shifts* is generated, as specified in the body of (8). For example, chasing (8) with the last tuple in Table 3, generates the new tuple $\langle \texttt{W1}, \texttt{Sep/9}, \texttt{Mark}, \bot \rangle$ in Table 4, with a fresh null for the shift (similarly for *W2*). This allows us to answer the query about the dates *Mark* works in *W1*: $\mathcal{Q}'(d)\colon\ \exists s \mathit{Shifts}(\texttt{W1}, d, \texttt{Mark}, s)$. We obtain *Sep/9*.

[4] If we have that relation, as in Example 1, then (6) could be replaced by a "static", non-navigational FD. This issue is further discussed in [30, appendix B].

Instead, the join between *PatientWard* and *UnitWard* in (7) enables upward-dimension navigation; and generates only one tuple for *PatientUnit* from each tuple in *PatientWard*, because each *Ward* member has only one *Unit* parent. ■

5 Properties of MD Datalog$^\pm$ Ontologies

Here, we first establish the membership of our MD ontologies, $\mathcal{M}$ (cf. Section 4) of a class of the Datalog± family. Membership is determined by the set $\Sigma_\mathcal{M}$ of its *tgds*. Next, we analyze the role of the constraints in $\kappa_\mathcal{M}$, in particular, of the set $\epsilon_\mathcal{M}$ of *egds*.

Proposition 1. MD ontologies are weakly-sticky Datalog± programs. ■

The proof (as other proofs) and a review of *weakly-sticky* Datalog± [12] can be found in the extended version [30, appendix A.]. A consequence of this result is that conjunctive query answering (QA) from $\Sigma_\mathcal{M}$ is in polynomial-time in data [12]. The complexity stays the same if we add negative constraints, *NCs*, of the forms (1) and (3), because they can be checked through the conjunctive queries in their bodies [12]. However, combining the *egds* in $\epsilon_\mathcal{M}$ with $\Sigma_\mathcal{M}$ could change things, and, in principle, even lead to undecidability of QA [7].

Example 7. Consider $\mathcal{I} = \{Surgery(\mathtt{W1}, \mathtt{John})\}$ and a weakly-sticky set Σ_T of *tgds*: $\sigma_1 : \exists z\ Surgeon(w, z) \leftarrow Surgery(w, p)$; $\sigma_2 : \exists y\ Assist(w, y) \leftarrow Surgery(w, p)$; $\sigma_3 : \exists z\ Surgery(z, x) \leftarrow Assist(w, x), Surgeon(w', x)$. Here, $chase(\mathcal{I}, \Sigma_T) = \{Surgery(\mathtt{W1}, \mathtt{John}), Assist(\mathtt{W1}, \perp_1), Surgeon(\mathtt{W1}, \perp_2)\}$.

Now, if we add the *egd* ε: $y = z \leftarrow Assist(w, z), Surgeon(w, y)$, the chase is infinite: $chase(\mathcal{I}, \Sigma_T \cup \{\varepsilon\}) = \{Surgery(\mathtt{W1}, \mathtt{John}), Assist(\mathtt{W1}, \perp_1), Surgeon(\mathtt{W1}, \perp_1), Surgery(\perp_2, \perp_1), Assist(\perp_2, \perp_3), Surgeon(\perp_2, \perp_3), Surgery(\perp_4, \perp_3), \ldots\}$.

These non-failing chases give different answers to the boolean conjunctive query (BCQ) $\mathcal{Q}$: $\exists w x w'(Assist(w; x) \wedge Surgeon(w'; x))$: $chase(\mathcal{I}, \Sigma_T \cup \{\varepsilon\}) \models \mathcal{Q}$, but $chase(\mathcal{I}, \Sigma_T) \not\models \mathcal{Q}$. ■

This example shows a harmful interaction between the *tgds* and an *egd*. They infinitely fire each other, making infinite an initially finite chase. The interaction also has an effect on QA. A *separability condition* on the combination of *egds* and *tgds* guarantees a harmless interaction wrt. QA.

Definition 1. [11, 14] Let Σ be formed by a set Σ_T of *tgds* and a set Σ_E of *egds*. Σ_E and Σ_T are *separable* if, for every instance $\mathcal{I}$ for which the chase of Σ on $\mathcal{I}$ does not fail, and BCQ $\mathcal{Q}$, $chase(\mathcal{I}, \Sigma) \models \mathcal{Q}$ if and only if $chase(\mathcal{I}, \Sigma_T) \models \mathcal{Q}$. ■

Example 7 shows a case of non-separability. Separability tells us that we can safely ignore Σ_E for QA. More precisely, if separability holds and QA is decidable under the *tgds*, then it is also decidable under the combination of *tgds* and *egds* : (a) (combined) chase failure can be decided by posing conjunctive queries associated to the bodies of the *egds* [14, theo. 1]; (b) if it does not fail, QA can be done with the *tgds* alone. Even more, under separability, the complexity of QA on $\mathcal{I} \cup \Sigma$ is the same as for $\mathcal{I} \cup \Sigma_T$ [11, 13, 14].

Proposition 2. For an MD ontology $\mathcal{M}$ with a set $\Sigma_{\mathcal{M}}$ of *tgds* as in (4) and set $\epsilon_{\mathcal{M}}$ of *egds* as in (2), separability holds if, for every *egd* in $\epsilon_{\mathcal{M}}$, the variables in the equality (in the head) occur in categorical positions in the body. ■

In combination with Proposition 1, we obtain:

Corollary 1. Under the hypothesis of Proposition 2, QA from an MD ontology can be done in polynomial-time in data. ■

Under the hypothesis of Proposition 2, our MD ontologies are separable and enjoy the good properties we just mentioned. However, some good properties can still be preserved with non-separable MD ontologies. The next example motivates this result.

Example 8. (ex. 7 cont.) Let us modify our ontology. Now, $\Sigma'_T = \{\sigma_1, \sigma_2\}$, and the *egd* is still ε. Now, both chases are finite: $chase(\mathcal{I}, \Sigma'_T \cup \{\varepsilon\}) = \{Surgery(\texttt{W1}; \texttt{John}),\ Assist(\texttt{W1}; \bot_1),\ Surgeon(\texttt{W1}; \bot_1)\}$; and $chase(\mathcal{I}, \Sigma'_T) = \{Surgery(\texttt{W1}; \texttt{John}),\ Assist(\texttt{W1}; \bot_1),\ Surgeon(\texttt{W1}; \bot_2)\}$. (As before, we use ";" to separate categorical from non-categorical attributes.) The *egd* is not separable from the *tgds*. Actually, for the same query $\mathcal{Q}$ of Example 7, and the non-failing chases, it holds: $chase(\mathcal{I}, \Sigma'_T \cup \{\varepsilon\}) \models \mathcal{Q}$, but $chase(\mathcal{I}, \Sigma'_T) \not\models \mathcal{Q}$. ■

In this example, despite the lack of separability, the application of *egds* does not trigger new *tgds* during the chase (as happens in Example 7). This is due (cf. Lemma 1 below) to the fact that $\Sigma'_T \cup \{\varepsilon\}$ respects a condition imposed on our MD ontologies: joins in *tgd* bodies only between categorical attributes. (The ontology in Example 7 had σ_3, which violates this condition.) Lemma 1 below tells us that with MD ontologies, applying *egd* chase steps does not increase the number of *tgd* chase steps.[5]

Lemma 1. For an MD ontology $\mathcal{M}$ with a set $\Sigma_{\mathcal{M}}$ of *tgds* as in (4) and a set $\epsilon_{\mathcal{M}}$ of *egds* as in (2), applying an *egd* chase step does not cause any new application of a ground *tgd*, i.e. a *tgd* body ground instantiation that did not appear without the *egds*. ■

With weakly-sticky sets of *tgds* the chase may not terminate, due to an infinite number of *tgd* chase steps. This is in particular the case for the set of *tgds* in our MD ontologies. However, QA on weakly-sticky *tgds* can be done in polynomial-time by querying an initial portion of the chase that has a polynomial *depth* [12]. By Lemma 1, if we add *egds*, QA can still be done by querying an initial portion of the chase (including *egds* now) that has the same (polynomial) *depth* as that for *tgds* alone. So, although *egds* in our MD ontologies may have an effect on QA (the two initial portions can be different), the complexity does not change wrt. to having only the *tgds*.

Proposition 3. For an MD ontology, QA is in polynomial-time in data. ■

6 MD Contexts for Quality Data

We now show in general how to use a MD context, $\mathfrak{C}$, containing MD ontologies for quality data specification and extraction wrt. a database instance D for schema $\mathcal{S}$. We will at the same time, for illustration and fixing ideas, revisit the example in Section 2, putting it in terms of the MD context elements we presented in Section 4. Context $\mathfrak{C}$, as shown in Fig. 4, contains:

[5] We assume the chase, after the enforcement of a (ground) *tgd*, applies all the *egds*.

1. Nickname predicates $R' \in \mathcal{S}'$ for predicates R of original schema $\mathcal{S}$. In this case, the R' have the same extensions as in D, producing a material or virtual instance D' within $\mathfrak{C}$.

For example, $\mathit{Measurements}' \in \mathcal{S}'$ is a nickname predicate for $\mathit{Measurements} \in \mathcal{S}$, whose initial contents (in D) is under quality assessment.

2. The *core MD ontology*, $\mathcal{M}$, that includes a partial instance, $\mathcal{D}_{\mathcal{M}}$, containing dimensional, categorical data; and the Datalog± ontology with *tgds* $\Sigma_{\mathcal{M}}$, and constraints $\kappa_{\mathcal{K}}$, among them, the *egds* $\epsilon_{\mathcal{M}}$ of Section 4. We assume that application dependent guidelines and constraints are all represented as components of $\mathcal{M}$.

In our running example, $\mathit{PatientUnit}$, $\mathit{PatientWard}$, $\mathit{WorkingSchedules}$ and $\mathit{WorkingTimes}$ are categorical relations. $\mathit{UnitWard}$, $\mathit{DayTime}$ are parent-child relations in the Hospital and Time dimensions, resp. The followings are dimensional rules (*tgds*) of $\Sigma_{\mathcal{M}}$: (with (9) a new version of (7) allowing upward-navigation in two dimensions)[6]

$$\mathit{WorkingTimes}(u,t;n,y) \leftarrow \mathit{WorkingSchedules}(u,d;n,y), \mathit{DayTime}(d,t).$$

$$\mathit{PatientUnit}(u,t;p) \leftarrow \mathit{PatientWard}(w,d;p), \mathit{DayTime}(d,t), \mathit{UnitWard}(u,w). \quad (9)$$

3. The set of *quality predicates*, $\mathcal{P}$, with their definitions in non-recursive Datalog (possibly with negation, not), in terms of categorical predicates in $\mathcal{R}$ and built-in predicates. They may have partial or full extensions in the contextual instance I (that includes $\mathcal{D}_{\mathcal{M}}$). A quality predicate reflects an application dependent specific quality concern.

Now, $\mathit{TakenByNurse}$ and $\mathit{TakenWithTherm}$ are quality predicates with definitions on top of $\mathcal{M}$, addressing quality concerns about the nurses and the thermometers:

$$\mathit{TakenByNurse}(t,p,n,y) \leftarrow \mathit{WorkingTimes}(u,t;n,y), \mathit{PatientUnit}(u,t;p). \quad (10)$$

$$\mathit{TakenWithTherm}(t,p,b) \leftarrow \mathit{PatientUnit}(u,t;p), u = \mathtt{Standard}, b = \mathtt{B1}. \quad (11)$$

Furthermore, and not strictly inside context $\mathfrak{C}$, there are predicates $R_1^q, ..., R_n^q \in \mathcal{S}^q$, the *quality versions* of $R_1, ..., R_n \in \mathcal{S}$. They are defined through *quality data extraction rules* written in non-recursive Datalog, in terms of nickname predicates (in $\mathcal{S}'$), categorical predicates (in $\mathcal{R}$), and the quality predicates (in $\mathcal{P}$), and built-in predicates. Their definitions (the α_i^q in Fig. 4) impose conditions corresponding to user's data quality profiles, and their extensions form the quality data (instance).

The quality version of $\mathit{Measurements}$ is $\mathit{Measurement}^q \in \mathcal{S}^q$, with the following definition, which captures the intended, clean contents of the former:

$$\begin{aligned}\mathit{Measurement}^q(t,p,v) \leftarrow\ & \mathit{Measurement}'(t,p,v), \mathit{TakenByNurse}(t,p,n,y), \quad (12)\\ & \mathit{TakenWithTherm}(t,p,b), b = \mathtt{B1}, y = \mathtt{certified}.\end{aligned}$$

Quality data can be obtained from the interaction between the original source D and the context $\mathfrak{C}$, in particular using the MD ontology $\mathcal{M}$. For that, queries have to be posed to the context, in terms of predicates S^q, the quality versions of those of D. A query could be as direct as asking, e.g. about the contents of predicate $\mathit{Measurement}^q$ above, or a conjunctive query involving predicates S^q.

A naive user -not familiar with the exact interaction with the context- who expects to obtain quality data from D will express a query $\mathcal{Q}$ is terms of the original schema

[6] A *tgd* may support multidimensional navigation and in multiple directions.

$\mathcal{S}$. However, the information system will rewrite the query into $\mathcal{Q}^q$, in terms of the predicates in $\mathcal{S}^q$. Consequently, the *quality answers* to $\mathcal{Q}$, are defined as those that are *certain* through the context:

Definition 2. For D an instance for schema $\mathcal{S}$, $\mathfrak{C}$ the context containing MD ontology $\mathcal{M}$, and definitions $\Sigma^{\mathcal{P}}, \Sigma^q$ of quality and quality version predicates, resp., the set of *clean answers* to a conjunctive query $\mathcal{Q}(\bar{x})$ on schema $\mathcal{S}$ is:

$$QAns_D^{\mathfrak{C}}(\mathcal{Q}) \;=\; \{\bar{c} \mid D \cup \mathcal{M} \cup \Sigma^{\mathcal{P}} \cup \Sigma^q \;\models\; \mathcal{Q}^q[\bar{c}]\}. \qquad \blacksquare$$

For example, this is the initial query asking for (quality) values for Tom Waits' temperature: $\mathcal{Q}(t,v)$: $\mathit{Measurements}(t, \texttt{Tom Waits}, v) \wedge \texttt{Sep5-11:45} \leq t \leq \texttt{Sep5-12:15}$, which, in order to be answered, has to be first rewritten into: $\mathcal{Q}^q(t,v)$: $\mathit{Measurements}^q(t, \texttt{Tom Waits}, v) \wedge \texttt{Sep5-11:45} \leq t \leq \texttt{Sep5-12:15}$.

To answer this query, first (12) can be used, obtaining a contextual query:

$$\begin{aligned}\mathcal{Q}^{\mathfrak{C}}(t,v)\colon\;\; & \mathit{Measurement}'(t,p,v) \wedge \mathit{TakenByNurse}(t,p,n,\texttt{certified}) \wedge \\ & \mathit{TakenWithTherm}(t,p,\texttt{B1}) \wedge p = \texttt{Tom Waits} \wedge \\ & \texttt{Sep/5-11:45} \leq t \leq \texttt{Sep/5-12:15}.\end{aligned}$$

This query will in turn, use the contents for $\mathit{Measurement}'$ coming from D, and the quality predicate definitions (10) and (11), eventually leading to a conjunctive query expressed in terms of $\mathit{Measurement}'$ and MD predicates only, namely:

$$\begin{aligned}\mathcal{Q}^{\mathcal{M}}(t,v)\colon\;\; & \mathit{Measurement}'(t,p,v) \wedge \mathit{WorkingTimes}(u,t;n,y) \wedge \\ & \mathit{PatientUnit}(u,t;p) \wedge u = \texttt{Standard} \wedge y = \texttt{certified} \wedge \\ & p = \texttt{Tom Waits} \wedge \texttt{Sep/5-11:45} \leq t \leq \texttt{Sep/5-12:15}.\end{aligned}$$

At this point, QA from a weakly-sticky ontology has to be performed. We know that this can be done in polynomial time in data. However, there is still a need for practical QA algorithms. Doing this goes beyond the scope of this paper. In [29] we describe some ideas on the development and optimization of such an algorithm.

7 Conclusions

Contexts, in particular, the multidimensional ones introduced in this work, allow us to specify data quality conditions, and to retrieve quality data. This is done by first mapping a data source, possibly with dirty data, into the context. The quality data can be materialized (possibly generating more than one intended clean instance) or be virtually defined. In both cases, it can be retrieved via queries. This latter idea of cleaning data on-the-fly is reminiscent of *consistent query answering* [5]. The main and important difference is that, instead of having (possibly violated) integrity constraints, with contexts we have a much more complex semantic framework for the definition of "repairs" (intended clean instances in our case) and consistent answers (the certain clean answers here).

There is still much to do in terms of development and optimization of practical query answering algorithms for weakly-sticky ontologies. Some first steps are reported in [29]. Implementation and experiments are matter of future work.

Several extensions of the current work have been or are being investigated. Those extensions can be found in the extended version of this paper [30, appendix B]. Some of them are as follows:

1. Uncertain downward-navigation when, *tgds* allow existentials on categorical attributes. A parent in a category may have multiple children in the next lower category. Under the assumption of complete categorical data, we know it is one of them, but not which one.

2. Our MD ontologies fully capture the taxonomy-based data model [26, 27] and its taxonomy relational algebra (TRA) for query answering. Our appraoch goes beyond [27] in the sense that, first, our categorical relations, by having non-categorical attributes, generalize t-relations. Secondly, the dimensional rules in our MD ontologies capture the TRA, and offer existential variables for handling incomplete data. Finally, we also include and support ontological constraints, such as NCs and *egds* for restricting dimension navigation.

3. The negative constraints (and *egds*, mainly in the separable case) can and are checked on the result of the chase. We think a more natural and practical approach would be to integrate constraint checking with data generation, restricting the latter process. This would amount to compiling constraints into *tgds*, which might lead to the use of negation in *tgd* bodies. This opens new problems. However, limited forms of negations have been introduced in Datalog$\pm$ [13].

4. We may relax the assumption on complete categorical data. This brings many new issues and problems that require investigation; from query answering to the maintenance of *structural semantic constraints*, such as strictness and homogeneity, on the HM model and our extension of it.

References

[1] A. Artale, D. Calvanese, R. Kontchakov and M. Zakharyaschev. The DL-Lite Family and Relations. *J. Artif. Intell. Res.*, 36, 2009, pp. 1-69.

[2] M. Alviano, W. Faber, N. Leone and M. Manna. Disjunctive Datalog with Existential Quantifiers: Semantics, Decidability, and Complexity Issues. *TPLP*, 2012, 12:701-718.

[3] C. Batini and M. Scannapieco. *Data Quality: Concepts, Methodologies and Techniques*. Springer, 2006.

[4] L. Bertossi, F. Rizzolo and J. Lei. Data Quality is Context Dependent. *Proc. VLDB WS on Enabling Real-Time Business Intelligence (BIRTE'10)*, Springer LNBIP 48, 2011, pp. 52-67.

[5] L. Bertossi. *Database Repairing and Consistent Query Answering*. Morgan & Claypool, 2011.

[6] C. Bolchini, E. Quintarelli and L. Tanca. CARVE: Context-Aware Automatic View Definition over Relational Databases. *Information Systems*, 2013, 38:45-67.

[7] A. Cali, D. Lembo and R. Rosati. On the Decidability and Complexity of Query Answering over Inconsistent and Incomplete Databases. *Proc. PODS*, 2003, pp. 260-271.

[8] A. Cali, G. Gottlob and T. Lukasiewicz. Datalog$^{\pm}$: A Unified Approach to Ontologies and Integrity Constraints. *Proc. ICDT*, 2009, pp. 14-30.

[9] A. Cali, G. Gottlob, T. Lukasiewicz, B. Marnette and A. Pieris. Datalog$^{\pm}$: A Family of Logical Knowledge Representation and Query Languages for New Applications. *Proc. LICS*, 2010, pp. 228-242.

[10] A. Cali, G. Gottlob and A. Pieris. Query Answering under Non-Guarded Rules in Datalog+/-. *Proc. RR*, 2010, pp. 1-17.

[11] A. Cali, G. Gottlob and A. Pieris. Ontological Query Answering under Expressive Entity-Relationship Schemata. *Information Systems*, 2012, 37(4):320-335.

[12] A. Cali, G. Gottlob and A. Pieris. Towards More Expressive Ontology Languages: The Query Answering Problem. *Artificial Intelligence*, 2012, 193:87-128.

[13] A. Cali, G. Gottlob and Th. Lukasiewicz. A General Datalog-Based Framework for Tractable Query Answering over Ontologies. *Journal of Web Semantics*, 2012, 14:57-83.

[14] A. Cali, M. Console, and R. Frosini. On Separability of Ontological Constraints. *Proc. AMW*, 2012, pp. 48-61.

[15] D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, A. Poggi, M. Rodriguez-Muro, R. Rosati, M. Ruzzi and D. F. Savo. The MASTRO System for Ontology-Based Data Access. *Semantic Web*. 2011, 2(1):43-53.

[16] E. Franconi and U. Sattler. A Data Warehouse Conceptual Data Model for Multidimensional Aggregation. *Proc. DMDW*, CEUR Proceedings, Vol. 19, 1999.

[17] E. Franconi and U. Sattler. A Data Warehouse Conceptual Data Model For Multidimensional Aggregation: A Preliminary Report. *AI*IA Notizie*, 1999, 1:9-21.

[18] G. Gottlob, G. Orsi and A. Pieris. Query Rewriting and Optimization for Ontological Databases. *ACM Trans. Database Syst.*, 2014, 39(3):25.

[19] R. Fagin, P. G. Kolaitis, R. J. Miller, and L. Popa. Data Exchange: Semantics and Query Answering. *Theoretical Computer Science*, 2005, 336:89-124.

[20] C. Hurtado, C. Gutierrez and A. Mendelzon. Capturing Summarizability with Integrity Constraints in OLAP. *ACM Transactions on Database Systems*, 2005, 30:854-886.

[21] T. Imielinski and W. Lipski Incomplete Information in Relational Databases. *Journal of the ACM*, 1984, 31(4):761-791.

[22] L. Jiang, A. Borgida and J. Mylopoulos. Towards a Compositional Semantic Account of Data Quality Attributes. *Proc. ER*, 2008, pp. 55-68.

[23] A. Maleki, L. Bertossi and F. Rizzolo. Multidimensional Contexts for Data Quality Assessment. *Proc. AMW*, 2012, CEUR Proceedings, Vol. 866, pp. 196-209.

[24] M. Lenzerini. Data Integration: A Theoretical Perspective. *Proc. PODS*, 2002, pp. 233-246.

[25] N. Leone, M. Manna, G. Terracina, and P. Veltri. Efficiently Computable Datalog$^\exists$ Programs. *Proc. KR*, 2012, pp. 1323.

[26] D. Martinenghi and R. Torlone. Querying Context-Aware Databases. *Proc. FQAS*, 2009, pp. 76-87.

[27] D. Martinenghi and R. Torlone. Taxonomy-Based Relaxation of Query Answering in Relational Databases. *The VLDB Journal*, 2014, 23(5):747-769.

[28] M. Milani, L. Bertossi and S. Ariyan. Extending Contexts with Ontologies for Multidimensional Data Quality Assessment. *Proc. 5th International Workshop on Data Engineering meets the Semantic Web (DESWeb)*. IEEE Data Engineering Workshops (ICDEW), 2014, pp. 242 - 247.

[29] M. Milani and L. Bertossi. Tractable Query Answering and Optimization for Extensions of Weakly-Sticky Datalog$\pm$. Submitted, under review, 2015.

[30] M. Milani and L. Bertossi. Ontology-Based Multidimensional Contexts with Applications to Quality Data Specification and Extraction. Extended version of this paper. http://people.scs.carleton.ca/~bertossi/papers/obmcExt.pdf

[31] R. Reiter. Towards a Logical Reconstruction of Relational Database Theory. In *On Conceptual Modelling*, M.L. Brodie, J. Mylopoulos and J.W. Schmidt (eds.), Springer, 1984, pp. 191-233.

A Appendix

A.1 Brief review of weakly sticky Datalog$\pm$

Sticky Datalog$^\pm$ programs are defined by means of a body variable *marking procedure* that takes as input the set Σ of *tgds*. It uses *positions*, that indicate arguments (or their numbers) in predicates, and has two steps:
(1) *Preliminary step*: for each $\sigma \in \Sigma$ and variable $x \in \mathit{body}(\sigma)$, if there is an atom $a \in \mathit{head}(\sigma)$ such that x does not appear in a, mark each occurrence of x in $\mathit{body}(\sigma)$.
(2) *Propagation step*: for each $\sigma \in \Sigma$, if a marked variable in $\mathit{body}(\sigma)$ appears at position p, then for every $\sigma' \in \Sigma$ (including σ), mark each occurrence of the variables in $\mathit{body}(\sigma')$ that appear in $\mathit{head}(\sigma')$ in the same position p.

Example 9. The following program shows the marked variables (underlined) after applying the preliminary step:

$$\begin{aligned} \mathit{Specialist}(d,x,n) &\leftarrow \mathit{Assist}(d,x), \mathit{Nurse}(d,n). \\ \mathit{Doctor}(x) &\leftarrow \mathit{Specialist}(\underline{d},x,\underline{n}). \end{aligned}$$

In the first rule, variables d and n are marked after applying one propagation step since they appear in head in marked positions only ($\mathit{Specialist}[1], \mathit{Specialist}[3]$), and the final, marked program is:

$$\begin{aligned} \mathit{Specialist}(d,x,n) &\leftarrow \mathit{Assist}(\underline{d},x), \mathit{Nurse}(\underline{d},\underline{n}). \\ \mathit{Doctor}(x) &\leftarrow \mathit{Specialist}(\underline{d},x,\underline{n}). \end{aligned}$$ ■

A set of *tgds* is *sticky* when, at the end of the marking procedure, there is no *tgd* with a marked variable in its body that occurs more than once. From Example 9, we can see that the program is *not* sticky since d in the first rule is marked and occurs twice in $\mathit{Assist}[1]$ and $\mathit{Nurse}[1]$.

The definition of *weakly-sticky* (WS) Datalog$^\pm$ programs appeals to conditions on repeated variables in *tgd* bodies, and is based on the notion of *dependency graph* and the *finite positions* in such a graph. More precisely, given a set of *tgds* Σ over schema $\mathcal{S}$, a directed *dependency graph* $G_\Sigma(V,E)$ is constructed. The vertices in V are positions of the predicates in $\mathcal{S}$, and the edges in E are defined as it follows. For every $\sigma \in \Sigma$ and non-existential variable x in $\mathit{head}(\sigma)$ and in position p in $\mathit{body}(\sigma)$: (1) for each occurrence of x in position p' in $\mathit{head}(\sigma)$, create an edge from p to p'; (2) for each existential variable z in position p'' in $\mathit{head}(\sigma)$, create a *special edge* from p to p''.

Example 10. Consider a set of *tgds* Σ:

$$\begin{aligned} \exists x\ \mathit{Assist}(x,n) &\leftarrow \mathit{Assist}(n,\underline{d}). \\ \mathit{Specialist}(d,x,n) &\leftarrow \mathit{Assist}(\underline{d},x), \mathit{Nurse}(\underline{d},\underline{n}). \\ \mathit{Doctor}(x) &\leftarrow \mathit{Specialist}(\underline{d},x,\underline{n}). \end{aligned}$$

First of all, the marked variables are the result of the marking procedure used to characterize the sticky programs (This is not required for the construction of the dependency graph, but will be needed to identify WS programs). The marking already shows that the program is *not* sticky (due to the doubly marked variable d in the body of the second rule).

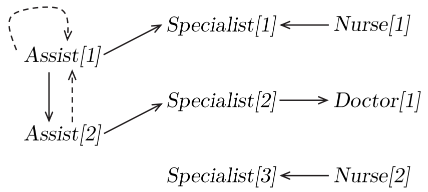


Fig. 5. Dependency graph for a set of *tgds*

Fig. 5 shows the dependency graph of Σ, where special edges are shown by dashed arcs. ■

The *rank of a position* is the maximum number of special edges over all (finite or infinite) paths ending at that position. Accordingly, $\Pi_F(\Sigma)$ denotes the set of positions of finite rank, and $\Pi_\infty(\Sigma)$ the set of positions of infinite rank. Intuitively, $\Pi_F(\Sigma)$ captures positions where finitely many values may appear during the chase; and $\Pi_\infty(\Sigma)$ those where infinitely many fresh null values may occur during the chase. A set of *tgds*, Σ, is *weakly-sticky* when, for every *tgd* and every variable in its body that occurs more that once, the variable is either non-marked or appears at least once in a position in $\Pi_F(\Sigma)$

Example 11. (example 10 cont.) According to the graph in Fig. 5, $\Pi_F(\Sigma)$ contains $\mathit{Specialist}[3]$, $\mathit{Nurse}[1]$ and $\mathit{Nurse}[2]$ and the other positions of the predicates in Σ are in $\Pi_\infty(\Sigma)$. The program in Example 10 is weakly-sticky. Notice that, the shared variable d in the second rule body is marked, but it appears at least once in a finite position, that is $\mathit{Nurse}[1]$. ■

A.2 Some Proofs

Proof of Proposition 1: We consider the set $\Sigma_\mathcal{M}$ of dimensional rules of the form (4). To prove the weak-stickiness, it suffices to show that every variable that occurs more than once in the rule body, appears at least once in a finite position ($\Pi_F(\Sigma_\mathcal{M})$). Due to the syntactic restrictions on the *tgds* in (4), repeated variables are allowed only in categorical positions (i.e. for categorical attributes). So, it is good enough to verify that categorical positions are in $\Pi_F(\Sigma)$.

Actually, in a rule of the form (4), a variable appears either in categorical positions or in non-categorical positions (but not both). Furthermore, special edges can only end with non-categorical positions, because existential variables are allowed only in those positions. Therefore, in the dependency graph of $\Sigma_\mathcal{M}$, although there might be a path from a categorical position to a non-categorical position (by a special edge), there is no path from a non-categorical position to a categorical position. This establishes that there is no path ending with a categorical position that includes a special edge. Therefore, every categorical position is in $\Pi_F(\Sigma)$. (In fact, every categorical position has zero rank.) ■

Proof of Proposition 2: We have to show that, for every instance $D_{\mathcal{M}}$, if $chase(D_{\mathcal{M}}, \Sigma_{\mathcal{M}})$ does not fail, then $chase(D_{\mathcal{M}}, \Sigma_{\mathcal{M}} \cup \epsilon_{\mathcal{M}}) \models \mathcal{Q}$ if and only if $chase(D_{\mathcal{M}}, \Sigma_{\mathcal{M}}) \models \mathcal{Q}$. For this, notice that our *egds* never equate two nulls or a null with a constant. This is because the equality is on variables that appear in categorical positions, and no null value will appear in them during the chase (cf. the proof of Proposition 1). Therefore, there is an applicable *egd* iff the chase fails (by equating two constants). So, either the chase fails, in which case the result is immediate, or it does not fail, but the *egds* have no effect on QA ■

.

Proof of Lemma 1: Let's assume there is an applicable *egd* $\sigma \in \epsilon_{\mathcal{M}}$ during the chase procedure. Notice that, σ equates either two categorical variables or two non-categorical variables. We assume they are non-categorical because equating categorical variables causes failure (there is no value invention in categorical positions) and the proof is immediate. Since, the shared variables in *tgds* of $\Sigma_{\mathcal{M}}$ as in (4) are categorical, applying σ does not change the applicability of any *tgd* in $\Sigma_{\mathcal{M}}$. ■

B Appendix: Discussion and Extensions

In this section we describe several extensions of the material presented in the main body of the paper. We also point to further developments are are matter of ongoing and future research. In this section we still make the assumption that categorical data are complete. We relax this assumption only in the final subsection, Section B.6.

B.1 Uncertain downward-navigation

Table 5. *DischargePatients*

	Inst.	Day	Patient
1	H1	Sep/9	Tom Waits
2	H1	Sep/6	Lou Reed
3	H2	Oct/5	Elvis Costello

General dimensional *tgds* of the form (4) restrict existential variables to non-categorical attributes. Under complete categorical data -the assumption we have made so far- we might want existentials on categorical attributes. These existentials are not relevant for upward navigation since every category member has a unique parent at the next upper level. But, downward navigation may find multiple children for category members, which creates *uncertainty* about the categorical values the existential values are associated to.

Example 12. (ex. 1 cont.) Consider an additional categorical relation *DischargePatients* (Table 5), about patients leaving an institution. Since patient was in a unit, we expect *DischargePatient* to generate data for *PatientUnit*, at the *Unit* level, down from the *Institution* level. For example, through a *tgd* such as:

$$\exists u\, InstitutionUnit(i, u), PatientUnit(u, d; p) \leftarrow DischargePatients(i, d; p). \quad (13)$$

Due to the existential on *Unit*, this *tgd* is not of the form (4) (the conjunction in the head can be eliminated with extra rules). More importantly, it involves a choice from the finitely many and fixed members of *Unit* associated to a member of *Institution*. The uniqueness of this member of the *Unit* category can be implied by an *egd* as follows:

$$u = u' \leftarrow \mathit{PatientUnit}(u, d; p), \mathit{PatientUnit}(u', d; p). \qquad \blacksquare$$

The existential variable in (13) represents a choice between the child elements while navigating downward from a parent element. This is fundamentally different from usual existential variables as in (4) that express value invention. The existential variable in (13) can be captured by the following rules, as one of them uses disjunction for uncertainty:

$$[\mathit{PatientUnit}(\texttt{Standard}, \boldsymbol{d}; p) \vee \mathit{PatientUnit}(\texttt{Intensive}, \boldsymbol{d}; p)] \leftarrow \mathit{DischargePatients}(\texttt{H1}, \boldsymbol{d}; p). \quad (14)$$

$$\mathit{PatientUnit}(\texttt{Terminal}, \boldsymbol{d}; p) \leftarrow \mathit{DischargePatients}(\texttt{H2}, \boldsymbol{d}; p). \quad (15)$$

As these partially grounded rules show, uncertainty only appears in (14) when we have multiple child elements (`Standard` and `Intensive`) with a parent element (`H1`). There is no uncertainly in (15) that navigates downward from a single parent (`H2`) to its child (`Terminal`). Clearly, this partial grounding depends on categorical data. Alternatively, we can employ a new data-independent syntactic form to allow a different kind of existential variable (e.g. u) that extends over existing values of a unary predicate, $\mathit{Unit}(\cdot)$:

$$\exists u^{[\vee_{\mathit{Unit}}]}\ \mathit{InstitutionUnit}(i, u), \mathit{PatientUnit}(u, d; p) \leftarrow \mathit{DischargePatients}(i, d; p).$$

This idea is similar to nominals in DL [1] where each individual (e.g. `Standard`, `Intensive` and `Terminal`) in a concept (e.g. Unit), itself is treated as a new concept.

The semantics of such rules can be defined as disjunctive Datalog programs with existential variables (Datalog$^{\exists,\vee}$ [2]). That is extending universal models with the notion of universal model sets. They allows multiple universal models each of which captures a choice between the elements in the head of the rules. As a future work, we intend to study the complexity of QA on an MD ontology enhanced with these rules. We also want to extend existing QA algorithms [29] on weakly-sticky ontologies to be used for an enhanced MD ontology with these rules. This possibly involves changing the chase procedure to make choices that results to generating multiple chase instances.

B.2 Categorical keys

In Section 4 we did not make the assumption that the combination of categorical attributes in categorical relations for a key for them. Notice that this is usually the case in fact tables in DWHs. Making this assumption would add extra *egds* to our MD ontologies. According to Lemma 1, such *egds* do not increase the complexity of query answering. The categorical keys assumption simplifies the MD ontology in the sense that the chase for such an MD ontology always terminates. That is due to finitely many possible key combinations which in turn follows from our assumption about complete categorical data.

B.3 Interaction of quality predicate definitions and MD ontologies

As we saw in Section 6, quality predicates (and quality version predicates) are defined on top of the weakly-sticky MD ontology $\mathcal{M}$. The following example shows that the combination may lead to a non-weakly-sticky ontology, even when the extra definitions are in plain Datalog.

Example 13. Consider (8) and the following *tgd* in $\Sigma_{\mathcal{M}}$ for categorical relations Shifts, $\mathit{PatientWard}$, and $\mathit{WorkingSchedules}$ from Section 2:

$$\exists n' \exists t \ \mathit{WorkingSchedules}(u, d; n', t) \leftarrow \mathit{Shifts}(w, d; n, z),\ \mathit{UnitWard}(u, w).$$

Let us add a definition of a quality predicate $\mathit{PreferredUnits}(\mathit{Unit})$:

$$\sigma: \quad \mathit{PreferredUnits}(u) \leftarrow \ \mathit{Shifts}(w, d; n, z),\ \mathit{WorkingSchedules}(u, d; n, t), \\ z = \texttt{morning}, t = \texttt{certified}.$$

Although the *tgds* above for a weakly-sticky ontology, the combination with σ breaks this property. That is because $\mathit{Shifts}[3]$ and $\mathit{WorkingSchedules}[3]$ are positions in $\Pi_{\infty}(\Sigma_{\mathcal{M}} \cup \{\sigma\})$, and n in σ's body is repeated in only marked positions. ■

This kind of loss of weakly-stickiness is not a problem. The quality predicate extensions (or parts thereof) can still be computed as conjunctive queries on a weakly-sticky ontology, adding an extra, upper layer to the ontology that can be computed in polynomial time. QA can be done via quality predicate unfolding as a first step, as illustrated in Section 6.

B.4 Navigational vs. static constraints

In Section 4, we introduced the general syntactic forms of (2) and (3) for *egds* and NCs. These syntactic forms impose some semantic constraints while performing dimension navigation in their bodies. They are also general enough to represent navigation free static *egds* and negative constraints that can encode general integrity constraints such as functional dependencies and key constraints.

In Example 14, we show that, in principle, the dimension navigation can be separated to be done in a preliminary phase using a rule of the form 4. Then, the semantic constraints are imposed on the result after navigation is done.

Example 14. Consider the *egd* of (6). We can split it into a dimensional rule of the form (4) and a static *egd* as follows:

$$\mathit{ThermTemp}(u, t; n) \leftarrow \mathit{Therm}(w, t; n),\ \mathit{UnitWard}(u, w).$$
$$t = t' \leftarrow \mathit{ThermTemp}(u, t; n),\ \mathit{ThermTemp}(u, t'; n').$$

Similarly, for (5), the following dimensional rule and static NC imply the same constraint:

$$\mathit{PatientTemp}(u, m; p) \leftarrow \mathit{PatientWard}(w, d; p), \mathit{UnitWard}(u, w), \mathit{MonthDay}(m, d).$$
$$\bot \leftarrow \mathit{PatientTemp}(\texttt{Intensive}, \texttt{August/2005}; p).$$ ■

The flexible syntactic forms of (2) and (3) are particularly important since they impose constraints while performing navigation. As a result, they remove some unnecessary chase steps during the chase procedure by avoiding additional predicates (e.g. *ThermTemp* and *PatientTemp*).

The negative constraints (and *egds* , mainly in the separable case) can and are checked on the result of the chase. We think a more natural and practical approach would be to integrate constraint checking with data generation, restricting the latter process. This would amount to compiling constraints into *tgds*, which might lead to the use of negation in *tgd* bodies. This opens new problems. However, limited forms of negations have been introduced in Datalog$\pm$ [13].

B.5 Datalog$^\pm$ and taxonomy-based queries

In this section, we propose a reconstruction of *taxonomy-based data model* [27] using the MD ontology. The main elements of this data model are *taxonomy*, *t-relations (taxonomy-relations)* and *TRA (taxonomy-based relational algebra)* for QA on t-relations. First, we shortly review the taxonomy-based model and then show how it can be captured by the MD ontology.

A *taxonomy* T is a set of *h-domains (hierarchical domains)*. An *h-domain* is a set of levels with a partial order between them (similar to dimensions in HM data model, where levels are categories). A level has a set of members. An h-domain and its levels must satisfy some basic conditions, similar to dimensions in HM data model , to be a valid h-domain (cf. [27]).

A *t-relation schema (t-schema)* is defined over a taxonomy T by $R = (C_1 : l1, ..., C_k : l_k)$ where R is relation name, each C_i is a distinct attribute name and each l_i is a level of some h-domain in T. A *t-tuple* t over a t-schema $R = (C_1 : l_1, ..., C_k : l_k)$ for a taxonomy T is a function mapping each attribute C_i to a member of l_i . A *t-relation* over R is a set of t-tuples over R.

A t-relation is represented in an MD ontology by a categorical relation. Let $R(C_1 : l1, ..., C_k : l_k)$ be a t-schema. The t-relation is captured in the MD ontology by a categorical relation with the schema, $R(C_1, ..., C_n;)$ (without non-categorical attribute). Each categorical attribute C_i replacing an attributes C_i of t-relation has a category that correspond to the level l_i.

Example 15. PatientWard in Example 1 can be represented in the taxonomy-aware data model as a t-relation with schema, $PatientWard = (ward : Ward, day : Day, patient : Patients)$ where *Ward*, *Day* and *Patient* are levels from the Hospital, Time and Patient h-domains. ■

Taxonomy-based relational algebra (TRA) is an extension of relational algebra over t-relations. It includes standard operations such as *selection*, σ, *projection*, π *and natural join*, $\bowtie$ and two new operators, *upward extension* and *downward extension*.

Consider R to be a t-relation with schema $R(C_1 : l1, ..., C_k : l_k)$, C be a contextual attribute in $\{C_1, ..., C_k\}$ defined over a level l, and l' be a level such that $l \leq_L l'$ ($\leq_L$ is the partial order relation between levels, as $\nearrow^*$ in HM data model). The *upward extension of* R *on* l', denoted by $\hat{\varepsilon}_{C:l}^{C':l'}(R)$, is a t-relation over $R'(C_1 : l1, ..., C_k : l_k, C' : l')$.

For every tuple $t = (c_1, ..., c_k)$ in t-relation R there is a tuple $t' = (c_1, ..., c_k, c')$ in R' such that $c' = CMAP_l^{l'}(c)$ st. $CMAP_l^{l'}$ is the roll-up mapping from members if l to l'.

Similarly, the *downward extension of* R *on* l' referred to as $\hat{\varepsilon}_{C':l'}^{C:l}(R)$ is defined as a t-relation over $R'(C_1 : l1, ..., C_k : l_k, C' : l')$, where for every tuple $t = (c_1, ..., c_k)$ in R there are tuples $t' = (c_1, ..., c_k, c')$ in R' such that $c' = CMAP_{l'}^{l}(c)$.

Example 16. Consider *PatientWard* as a t-relation. The upward extension of *PatientWard* on the *unit* attribute is a t-relation $\hat{\varepsilon}_{ward}^{unit}(PatientWard)$ with schema $(ward : Ward, day : Day, patient : Patient, unit : Unit)$. ■

The operators of TRA that are inherited from relation algebra (projection, selection and natural join) are naturally supported by Datalog$^\pm$. Here, we show that upward and downward extensions can also be expressed in an MD ontology. For a t-relation $R(C_1 : l1, ..., C_k : l_k)$ captured in the ontology by a categorical relation $R(C_1, ..., C_k;)$, the result of an upward extension $\hat{\varepsilon}_{C':l'}^{C:l}(R)$ $(C \in \{C_1, ..., C_k\})$ is expressed in the ontology as a new categorical relation with schema $R'(C_1, ..., C_k, C')$ such that the new categorical attribute is from the category/level l'. The data for this new categorical relation is generated by the following dimensional rule of the general form (4) (underlines variables participate in up/down extension):

$$R'(e_1, ..., \underline{e}, ..., e_n, \underline{e'}) \leftarrow [R(e_1, ..., \underline{e}, ..., e_n), \\ D_1(\underline{e'}, m_1), D_2(m_1, m_2), ..., D_k(m_k, \underline{e})].$$

A similar dimensional rule performs downward extension, $\hat{\varepsilon}_{C:l}^{C':l'}(R)$:

$$R'(e_1, ..., \underline{e}, ..., e_n, \underline{e'}) \leftarrow [R(e_1, ..., \underline{e}, ..., e_n), \\ D_1(\underline{e}, m_1), D_2(m_1, m_2), ..., D_k(m_k, \underline{e'})].$$

Example 17. Consider the *PatientWard* as a t-relation in Example 15 that corresponds to the *PatientWard* categorical relation in the proposed MD ontology. An upward extension operator, $\hat{\varepsilon}_{ward}^{unit}(PatientWard)$ is a new categorical relations $PatientWardUnit$ defined as follows:

$$PatientWardUnit(\underline{w}, t, p, \underline{u}) \leftarrow PatientWard(\underline{w}, t, p),\ UnitWard(\underline{u}, \underline{w}).$$ ■

The taxonomy-based data model is motivated by an earlier work in [26] that introduces a *context-aware data model*. The same results about capturing taxonomy-based data model in a MD ontology applies to context-aware data model.

B.6 Incomplete categorical data

So far in this work we have made the assumption that categorical data, i.e. in categories and categorical attributes, are complete. But we might want to consider incomplete categorical data, and we could have *tgds* that generate categorical data, upwards or downwards. The latter, in particular for dealing with uncertainty in downward navigation. This could be the case in Example 12, specifically in (13), we could assume the existential variable invents new elements in categorical positions. In the following, we discuss several issues in relation to the relaxation of this complete categorical data assumption, and of the corresponding value invention restriction.

B.6.1 Dimension modification. The immediate result of incomplete categorical data and element invention is that referential constraints of the form (1) may not hold, and the ontology becomes inconsistent. Therefore, we need to replace the referential constraints of the form (1) by rules that propagate invented values as new elements in categories: $K(e) \leftarrow R_i(\boldsymbol{\bar{e}_i}; \bar{a}_i)$. These new elements in categories might also trigger a sequence of changes to preserve the structure of dimension structure (cf. Section B.6.4).

B.6.2 Weakly-stickiness of the MD ontology. An important effect of incomplete categorical data is that Proposition 1 does not necessarily hold (the MD ontology is not generally weakly-sticky). The existential variables in categorical positions can create infinitely many new elements in these positions. As a result, there is possibility of repeated marked variables in these positions that violates weakly-stickiness.

Interestingly, adding categorical keys (Section B.2) while having incomplete categorical data, we can still preserve not only weakly-stickiness but also chase termination mentioned in Section B.2. This is basically due to the fact that element invention only happens while navigating downward and no new key is generated while doing upward navigation. Therefore still finitely many values can appear in categorical positions which proves repeated marked variables are always in these finite positions and so weakly-stickiness holds. The chase still terminates since there are finitely many combination of keys (some of them include new elements).

B.6.3 Adding *egds* and separability. These new elements also have effect on Lemma 1 and Proposition 2 since they both assume fixed elements. Lemma 1 does not necessarily hold since the equality in the heads of *egds* may equate two categorical values. That can trigger a join in the body of a tgd rule making new applicable tgds steps. Proposition 2 is not valid either since, even if we restrict equalities to categorical variables, they involve null values and equating them can break separability.

Also notice that, assuming incomplete categorical data, Lemma 1 does not generally hold. Consequently, other restriction must be imposed to make sure adding the extra *egds* of categorical keys does not increase complexity of QA over the ontology (i.e. non-conflicting keys [7, 12]).

B.6.4 Taking care of MD structural semantic constraints. Several "structural semantic constraints" have been proposed for HM dimensions (cf. Section 3). Among them, *strictness* and *homogeneity* are the most important since they ensure the *summarizability property* of the dimension, which, in essence, guarantees that cube views can be reused to correctly compute cube views at higher levels of the lattice [20].

Strictness requires that, for every elements e_1, e_2, e_3 and category c, if $e_1 <^* e_2$, $e_1 <^* e_3$, $\sigma(e_2) = c$ and $\sigma(e_3) = c$, then $e_2 = e_3$. Homogeneity requires that, for every element e and categories c, c', if $\sigma(e) = c$, and $c \nearrow^* c'$, then there is an element e' such that $\sigma(e') = c'$ and $e <^* e'$. These two properties together .

When we make the assumption that categorical data is complete, we can make sure before anything that the MD semantic constrains above are satisfied. Whatever we do next in terms of data generation will not harm them. However, this may change if we have incomplete categorical data and data is generated. We may want the MD semantic constraints to be kept satisfied.

To express the *strictness* property, we define new predicates in the MD ontology to represent $<$ and its transitive closure, $<^*$. For a dimension D, we refer to the new predicate (its transitive closure) as $D(.,.)$ ($D^*(.,.)$) which defined by rules of the following form (D_i is a parent-child predicate in dimension D): $D(x, y) \leftarrow D_i(x, y)$.

In the Hospital dimension, we represent $<_{Hospital}$ by $Hospital(.,.)$ defined by the following rules: $Hospital(e, e') \leftarrow UnitWard(e, e')$ and $Hospital(e, e') \leftarrow InstitutionUnit(e, e')$ and $Hospital(e, e') \leftarrow AllInstitution(e, e')$.

Now, we can check the strictness using *egds* of the following form (K is a category in the dimension , D): $e_1 = e_2 \leftarrow D^*(e, e_1), D^*(e, e_2), K(e_1), K(e_2)$.

For example, the strictness property for the Hospital dimension is represented by the following constraints:

$$e_1 = e_2 \leftarrow Hospital^*(e, e_1), Hospital^*(e, e_2), Institution(e_1), Institution(e_2).$$
$$e_1 = e_2 \leftarrow Hospital^*(e, e_1), Hospital^*(e, e_2), Unit(e_1), Unit(e_2).$$
$$e_1 = e_2 \leftarrow Hospital^*(e, e_1), Hospital^*(e, e_2), Ward(e_1), Ward(e_2).$$

The homogeneity constraint is captured in an MD ontology by rules of this form: $\exists e' \; D^*(e, e'), K'(e') \leftarrow K(e)$. There will be a rule for every child category K and ancestor category K'. In the Hospital dimension, an example for these rules is the following:$\exists i \; Hospital^*(w, i), Institution(i) \leftarrow Ward(w)$.

Note that, such a rule must be evaluated not as a normal *tgd* rule (in *tgd* chase steps) but as a constraint. In the sense that, if the rule is applicable during a (restricted) *tgd* chase step, the chase procedure reports an inconsistency in the ontology instead of generating new null value to satisfy the rule.