Quality Evaluation and Improvement Framework for Database Schemas - Using Defect Taxonomies

Lemaitre, Jonathan; Hainaut, Jean-Luc

Published in:
CAISE 2011

Publication date:
2011

Document Version
Early version, also known as pre-print

Quality Evaluation and Improvement Framework for Database Schemas - Using Defect Taxonomies

Jonathan Lemaitre and Jean-Luc Hainaut

Laboratory of Database Application Engineering - PReCISE research Center
Faculty of Computer Science, University of Namur
Rue Grandgagnage 21 - B-5000 Namur, Belgium
{jle,jlh}@info.fundp.ac.be
http://www.fundp.ac.be/precise

Abstract. Just like any software artefact, database schemas can (or should) be evaluated against quality criteria such as understandability, expressiveness, maintainability and evolvability. Most quality evaluation approaches rely on global metrics counting simple pattern instances in schemas. Recently, we have developed a new approach based on the identification of semantic classes of definite patterns. The members of a class are proved to be semantically equivalent (through the use of semantics preserving transformations) but are assigned different quality scores according to each criteria. In this paper, we explore in more detail the concept of bad pattern by proposing an intuitive taxonomy of defective patterns together with, for each of them, a better alternative. We identify four main classes of defects, namely *complex constructs*, *redundant constructs*, *foreign constructs* and *irregular constructs*. For each of them, we develop some representative examples and we discuss ways of improvement against three quality criteria: *simplicity*, *expressiveness* and *evolvability*. This taxonomy makes it possible to apply the framework to quality assessment and improvement in a simple and intuitive way.

Keywords: Conceptual data schema, quality, schema improvement, schema evaluation, schema transformation.

1 Introduction

Modern engineering approaches to system development lead to methods in which modeling activities have become prominent, notably through the so-called *model driven engineering (MDE)* initiative. According to these methods, the design of a complex software system appears as a hierarchy of models, starting from the goal model down to the source code of the concrete artifacts. Models derive from each other through transformations that preserve some of their intrinsic properties, such as correctness, information capacity or performance. In addition, most models use components of other models. Through these derivation and use dependencies, a defect in a source model potentially propagates to many

dependent models. In such an interconnected model network, the quality of the whole system critically depends on the quality of each of its models.

When dealing with the quality of information systems, one has to pay particular attention to the database component because of the significant role of this component in the whole system. Typically, a database can be the data provider of thousands of programs. Any flaw in the schema of this database may cause data inconsistencies, program malfunction or, at best, program code complexity to compensate for this defect. The total cost to pay will be even higher: database and program evolution will prove to be more complex and risky since both sane, compensating and flawed components will need to evolve.

Although the MDE is often seen as a new approach in the software engineering community, it has been the standard way of developing databases since the seventies, where the three-level methodologies were designed and progressively applied. They are based on a hierarchy of *schemas* (the database name for *models*[1]), namely the conceptual, logical, physical schemas, the latter being translated into DDL code. Those three levels are called *abstraction levels*. In addition, a schema is expressed in a specification language, based on a definite paradigm (figure 1). Since logical and physical schemas mostly derive from the conceptual schema through semantics preserving transformations, ensuring the highest quality of conceptual schemas is particularly important.

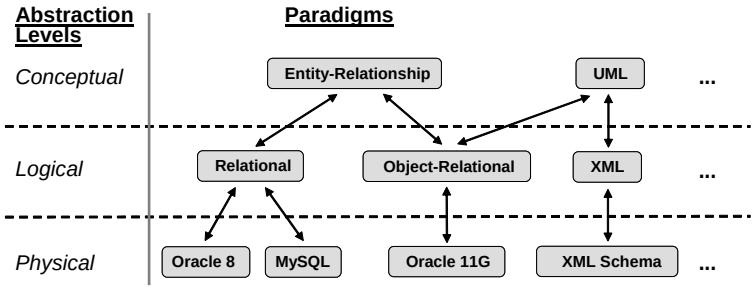

Fig. 1. A representation of the schema abstraction levels and some paradigms.

In [1], we proposed a database quality evaluation and improvement framework based on transformation techniques. Instead of computing global metrics for the whole schema, it first tries to identify semantically significant constructs that represent possible defects in schemas. Let us call *construct* an instance of a definite pattern comprising data structures and constraints [2]. An n-ary re-

[1] From this point, we will use the database terminology, i.e. a schema is the representation of the application domain and is expressed in a specification language called model (e.g. entity-relationship model, relational model).

[2] In the following discussion, for simplicity reason and where no ambiguity may arise, we will sometimes use the name *construct* to denote such a pattern as well as one of its instances

lationship type, an entity type without attributes, a series of attributes with similar names are all examples of constructs. A defect is a construct that is considered sub-optimal to translate the intention of the modeler. A *relationship entity type* (an entity type whose instances are used to connect instances of two other entity types) may be, in some circumstances, considered a defect since a mere many-to-many relationship type would better express the intention of the designer. Considering a set of about 20 basic conceptual patterns, we have defined as many equivalence classes, each of them gathering all the patterns that express the same semantics. For example, the relationship entity type and many-to-many relationship type patterns appear in the same equivalence class. In each class, we can identify its representative member, that is, the pattern that best expresses the common intention of the members of this class (and for this, called *best practice*). For example, the semantic pattern *many-to-many association* will be described by a class that includes, among a dozen equivalent patterns, the many-to-many relationship type, the relationship entity type, the multi-valued foreign key, the multi-valued embedded component. Clearly, the first pattern will be the representative member of this class. We will see in the section 3, that the best practice of an equivalence class depends on the quality criteria for the evaluation of which this class is used.

The qualification *defective* of a construct is not absolute [3] but depends on three factors, namely the abstraction level, the modeling paradigm and the quality criterion. For example, at the logical level, the *foreign key*, as the expression of a many-to-one relationship type, is optimal in a class of logical constructs but sub-optimal in a class of conceptual constructs. It is optimal in the SQL paradigm but not in the ADO Microsoft interface, based on a simple Entity-relationship model. It may be considered sub-optimal in an XML schema where element embedding may be preferred for performance reason.

In this paper, we will deepen the framework by exploring the space of conceptual defects and by attempting to classify them into an ontology of natural defect types. These reference defect types contribute to a better understanding of the third factor mentioned above: quality criterion. This classification will be used to improve our quality evaluation framework, but it has also been used in database design education [2] in the perspective of building high quality schemas.

Since most, if not all, database schemas include a certain amount of defects and considering that database design mainly is a creative task, we can expect the catalog of schema defect types being very large. In the following sections, we will concentrate on defects that degrade otherwise correct schemas. For example, a relational table that is not in 3NF is not intrinsically incorrect but it leads, among others, to expressiveness (two fact types are represented in the same table) and performance (space and update time) problems. The process of identifying these defects and improving their *structural* quality is generally known as *Conceptual normalization*.

The paper will be structured as follows. Section 2 presents a short state of the art in the role of defects in database schema quality. We recall the main

[3] For this reason, we have avoided the term *anti-pattern*

concepts of the framework in section 3. Section 4 describes the bases of quality analysis for conceptual schemas. In section 5, we present a taxonomy of conceptual data schema defects and discuss their improvement. The use of the framework extended by this taxonomy is presented in section 6. Section 7 concludes the paper.

2 State of the Art

Transformations are usually related to the functional requirements aspects of database schemas. Transforming a source schema must (should) preserve its information capacity [4] in such a way that the eventual DDL code completely translates the semantics of the conceptual schema. The use of transformations in the context of schema quality mainly concerns non-functional requirements, and, in this context, it has been rather limited. However, a few authors have already considered processes in which a local set of objects in a schema is replaced by another one in a way that improves some quality properties of the schema.

A first major (historical) proposal is the relational schema normalization process [4], based on functional dependencies mainly in order to remove redundancies at the data level. Though the term *transformation* was not used at that time, normalization decomposition actually makes use of semantics-preserving transformations[5]. These transformations can also be used to influence the performance of the database. Leaving the semantics of data unaltered but improving its redundancy or performance state, relational normalization clearly contribute to make the schema meet non-functional requirements. In [5], the authors studied the impact of relationships types attributes on the clarity of ER schemas. In a similar way in [6], Gemino and Wand have analysed the difference between the use of the mandatory and optional properties, also in ER schemas. Though these papers naturally called for substitution techniques to improve the readability of schemas, the authors did not push their analysis to this point.

Only a few authors have explicitly used semantics-preserving transformations for improving the schema quality. Among them, we can underline the framework of Assenova and Johanesson [7] for dealing with understandability of conceptual schemas. They assign qualitative quality scores to a set of transformations and propose to use them in order to improve schema quality. Rauh and Stickel [8] also use transformations in the context of conceptual schemas in order to normalize them and therefore to improve their quality.

The framework we propose is close to the work of Assenova and Johanesson [7]. Yet, we paid particular attention to genericity, referring to the possibility to use the framework on different abstraction levels, different paradigms and considering different quality criteria. Also, we did not associate quality preferences to the transformations themselves but to the structures.

[4] A discussion on semantics preserving transformation and information capacity can be found in [3].

[5] The concept of semantics preservation is a bit more complex in this context since data preservation and functional dependency preservation may conflict

3 Framework Reminder

In section 1, we introduced the main principles of our framework. In this section, we present some detail of the framework, based on reference [1], where the reader can find an extended description.

The framework is based on the use of semantics-preserving transformations and on the identification of specific structures in schemas. It relies on the fact that there are generally different ways to express a set of facts of the application domain, and that some of them are better than others according to definite criteria. In order to make it generic enough to deal with different data model, we use the GER model [9], a wide spectrum model that encompasses the main data models (ER, EER, Relational, UML, etc.) and allows to use object types that belong to different abstraction levels and paradigms in a single schema. The framework relies on four concepts: the **equivalence classes**, the **contexts**, the **ratings**, the **quality methods**.

An **equivalence class** EC_i is a set of constructs, i.e., $EC_i = \{C_{i1}, \ldots, C_{in}\}$, that all represent the same modeling intention. Moreover for any couple of distinct constructs C_{ij}, C_{ik} of EC_i, there exists a transformation sequence T composed of semantics preserving transformations such that $T(C_{ij}) = C_{ik}$. An equivalence class represents a common modeling intention which refers to a specific type of facts of the application domain. Similar constructs are defined as instances of a generic pattern, so that equivalence classes can be reduced to about 20 useful classes made up of patterns.

We illustrate the *non-functional* [6] *binary association* equivalence class in figure 2. It collects popular generic data structures intended to represent many-to-many associations between the members of two object classes. *(a)* is a many-to-many relationship type. *(b)* is a relationship entity type accompanied by two one-to-many relationship types. *(c)* and *(d)* represent associations through multivalued (c) and single-valued (d) foreign keys. Finally, *(e)* and *(f)* use two-way and one-way object references, borrowed from the object oriented data model.

A **context** describes the requirements against which the schema is evaluated. It is defined by a triple *(A,P,D)*, where A is an abstraction level, P is a data modeling paradigm and D is a set of quality requirements. The abstraction level A and the paradigm P define the use of a specific model, e.g., UML class diagrams at the logical level. According to level A, some constructs of P will become undesired. For example, a conceptual ER-like schema should not include constructs that explicitly or implicitly define a foreign key.

A **rating** is an ordering of the members of an equivalence class for a specific context. It states the extent to which each member meets the quality criteria D of the context. Different methods have been proposed to define these scores. Collecting expert estimation is the preferred technique since it requires less effort than standard empirical studies based on schema global evaluation. Ratings can be used in several ways, notably (1) to compute metrics for the schema under investigation and (2) to suggest schema improvement. A rating also allows to

[6] In the sense that it supports no functional dependencies

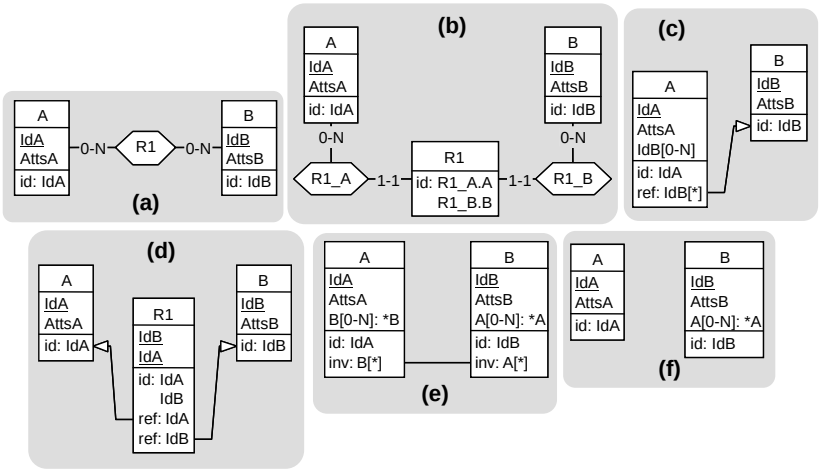

Fig. 2. *Non-functional association* equivalence class

identify in the equivalence class a *best practice* as the construct that has the highest score for the context of the rating.

A **quality method** comprises analysis, evaluation and improvement methods. An evaluation method provides global and detailed quality scores for the schema. An improvement method is based on the replacement of constructs with a low rating for a context by a better construct in its equivalence class, for instance the best practice of the context.

4 Quality Requirements

Quality has become a major research field in software engineering, though its scope, its objectives and its evaluation techniques have not gained sufficient consensus so far to consider it a mature domain. For example, similar but still significantly different definitions of the very basic concept of *understandability* can be found in [10], [11], [12] and [13]. Definitions have evolved with time and standards have been proposed such as the ISO quality standard [12, 14]. Unfortunately definitions available in a standard often appear too general and not intuitive enough when addressing the quality of a specific software product. This lack of precision also makes it uneasy to develop convincing operational methodologies and to build supporting tools.

In this paper, we consider three essential qualities of conceptual schema constructs, namely simplicity, expressiveness and evolvability. In the following, we provide definitions and interpretations of these non-functional requirements.

– **Simplicity:** Simplicity is a sub-characteristic of the understandability quality requirement in the ISO/IEC 9126 standard [12]. In [15], *a schema is said to be simple if it is constructed upon simple concepts*. The notion of *simple concept* relies on a measure of the complexity, which is itself related to

the number of some specific elements, such as relationship types. Instead we define the simplicity as the property of types of facts of the application domain being represented *as simply as possible*. This definition encompasses the notion of minimality and low complexity. A practical definition could be based on the number of objects and the nature of objects. A construct has a better minimality score than another one if it uses fewer objects to express the same fact type. We also consider the structural and cognitive complexity of the objects. For example, to represent a definite fact type, an elementary attribute is clearly less complex than a compound attribute or an entity type.

- **Expressiveness:** Like simplicity, expressiveness has a high impact on understandability in the ISO standard [12]. An early definition of expressiveness was suggested by Batini et al. in [16], where the authors defined it as *the richness of a schema*. In this paper, we follow the definition given in [15] inspired from the previous early definition, and more specifically the subconcepts of *concept expressiveness*, that measures whether *the concept* [the constructs] *of the schema are expressive enough to capture the main aspects of the reality*. We relate the expressiveness to the fact that a type of facts of the application domain is represented by a construct that naturally and clearly refers to its nature. For instance, the fact that *domestic appliances* form a variety of *products* should be represented by a subtype/supertype relation, provided the data model includes this type of constructs.
- **Evolvability:** We define evolvability as the ability of a construct to support possible changes in the application domain and to trigger as little impact as possible on the system artefacts that include (e.g., the schemas) and use (e.g., programs and HCI) this construct. While previous quality requirements address the understandability of the conceptual schema, this definition is adapted from the *changeability* definition of the ISO Quality standard [12]. Being able to adapt the schemas following new application domain changes is important, especially in the context of the MDE methods.

These three qualities synthesize some of the most important requirements for a database schema. In addition, they can be formally defined through our framework, in which they form the D component of a context. We will apply them to evaluate and illustrate the taxonomy of section 5. Besides, these qualities are not independent: a construct that increases the simplicity of a schema may lower its expressiveness. They can be considered separately or combined in order to reach a trade-off.

5 Defect Taxonomy

Teaching, modeling experience and schema analysis eventually allow to come up with a set of good modeling practices. The latter are the structures an experienced designer would most probably use to represent specific fact types of the application domain. We observe that many designers make other, sometimes unfortunate, choices, so that design flaws may often be found in database schemas.

In addition, the fact that a definite construct should be used is obviously context-dependent. Besides, a sound construct may not always meet all the requirements stated for some databases.

Based on experience and schema study, we progressively elaborated a set of constructs that appear to be poor design choices or defects in most situations. Among them, some can be replaced without altering the information capacity of the schema, that is, through the use of semantics preserving transformations. In such cases, there is a strong bond between the taxonomy and the framework since each couple of constructs (the *good* and the *bad*) belong to the same equivalence class. We gather these defective constructs into four categories, namely *complex*, *redundant*, *foreign* and *irregular* constructs.

We also analyse the impact of these defects on the three quality requirements defined in section 4 by comparing defects with the recommended alternative. Due to space limit, we discuss some defects only but we mention other popular constructs in each of the four categories. Additional description of defects can be found in chapter 17 of [2].

5.1 Complex Constructs

We qualify a construct *complex* when it is not the most straighforward way to represent the fact type of the application domain under consideration. So far, we have identified about a dozen complex construct classes. We detail three of them:

- **Attribute entity type** The construct represents a *property* of a concept of the application domain. It is made up of an entity type E_1 with one attribute A and that plays a *[1-1]* or *[1-N]* role in a binary relationship type R. The identifier of E_1 includes this attribute. It appears that the only goal of E_1 is to define a property of the concept represented by the other entity type in R. We call it E_0. E_0 and E_1 correspond respectively to the entity types *BOOK* and *KEYWORD* in figure 3 *(1.a)*. This goal would have been better achieved by a mere attribute. The suggested improvement would be to migrate A from E_1 to E_0 as illustrated in the figure 3 *(1.b)*. Considering the quality requirements we have defined, this transformation increases the simplicity and the expressiveness of the schema. Indeed, the new construct contains fewer objects and is closer to the application domain structure. On the contrary, the original construct favours evolvability. Indeed, should E_1 represent an important concept in the future, it could receive new attributes and play new roles without painful restructuring.
- **N-ary relationship type with a** $[0-1]$ **or** $[1-1]$ **role** As an identifier of the relationship type, this role is the functional determinant of each of the other roles. Therefore, the relationship type is decomposable into two binary functional relationship types as illustrated in figure 3 *(2)*. N-ary relationship types are intrinsically more complex and less intuitive than binary ones [7].

[7] as testified by the endless discussions on the semantics of N-ary associations in UML class diagrams!

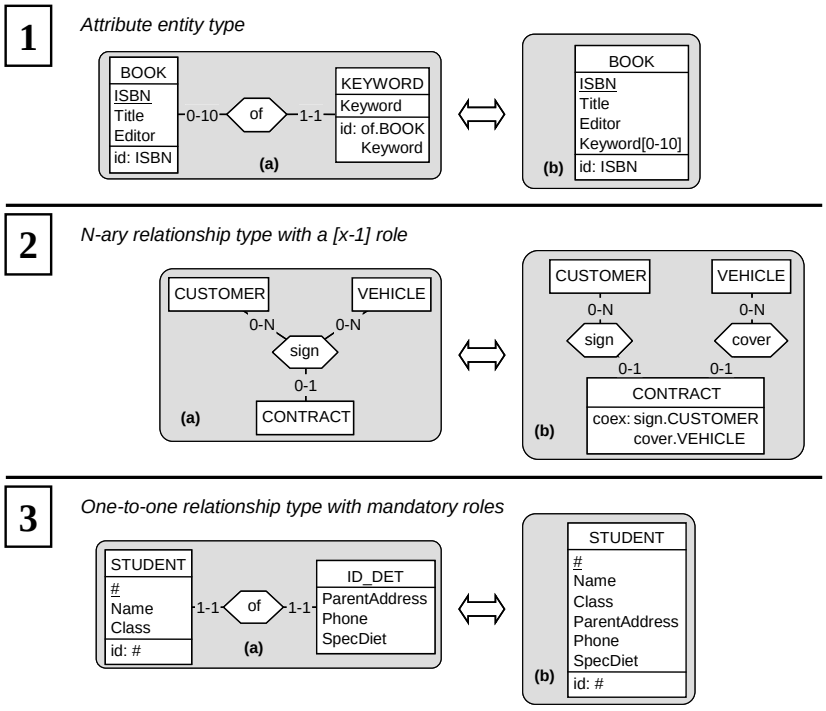

Fig. 3. Three complex constructs and their suggested alternatives.

In general, the decomposed pattern is simpler, more expressive and easier to evolve than the source construct.

– **One-to-one relationship type with mandatory roles** This pattern expresses a strong link between two concepts (no instance of a concept exists independently of an instance of the other one) (figure 3 *(3.a)*). In most situations, the latter just represent two aspects, or two complementary fragments, of the same concept. A better alternative (figure 3 *3.b*) can be to merge the entity types into a single entity type. This improves the simplicity and the expressiveness of the schema. Normally, evolvability should stay unchanged though, in some cases, it may decrease, depending on the semantics of each fragment.

The other complex constructs are classified in table 1, with the qualitative score difference when they are replaced by the alternative construct we suggest.

5.2 Redundant Contructs

A construct A is redundant with construct B when, in any database state, the instances of one of them can be computed from the instances of the other one. More complex situations may occcur (for instance where A and B are mutually

Table 1. Complex constructs: qualitative evaluation

	Simplicity	Expressiveness	Evolvability
Entity type			
ET with a weakly specified subtype → *supertype attribute*	+	-	-
ET with an empty subtype → *supertype attribute*	+	+	≈
Relationship entity types → *relationship type*	?	+	$\approx_+$
Implicit Is-A rel.: materialization → *explicit is-a*	+	+	$\approx_+$
Implicit Is-A rel.: upward inheritance → *explicit is-a*	-	+	$\approx_+$
Implicit Is-A rel.: downward inheritance → *explicit is-a*	≈	+	+
Relationship type			
Inter-attribute functional dependencies → *decomposition*	-	+	+
Attribute			
Complex attributes → *entity type*	-	+	+
One-component compound attribute → *desagregation*	+	+	?
Inter-ET functional dependencies → *decomposition*	-	+	+
Constraint			
Decomposed existence constraints → *merging*	+	≈	+

Legend: (-) Quality decrease (+) Quality improvement
(≈) Equivalent quality (?) Indeterminate quality change
($\approx_+$) Nearly equivalent quality with slight improvenent

dependent), but they will be ignored in the framework inasmuch as their solving requires expert knowledge. The correction of such defects consists in removing one of the source constructs (normally the lowest quality construct according to its ranking in its equivalence class), which is (trivially) a semantics-preserving transformation.

We classify the redundant constructs in two categories. The first category includes patterns in which some fact types of the application domain are expressed more than once. The **relationship/foreign key redundancy** (figure 4 *(a)*) pattern is an example of this category. This pattern comprises an attribute that references entities of another type (therefore acting as a foreign key) while a relationship type already expresses such relationships explicitly. These constructs are redundant and one of them must be removed. Since the foreign key suffers from another problem (it appears as a *foreign construct* at the conceptual level - see below), we remove it from the schema. A second example, not illustrated, is that of attribute *Amount* of entity type $ORDER$, the values of which can be computed from the values of attributes $OrderedQuantity$ of $DETAIL$ and $UnitPrice$ of $PRODUCT$. The derived objects should be removed from the schema in order to increase its simplicity and evolvability. The evaluation of ex-

pressiveness is somehow less intuitive. As we removed the less expressive objects, we consider that the expressiveness of the whole construct has increased too.

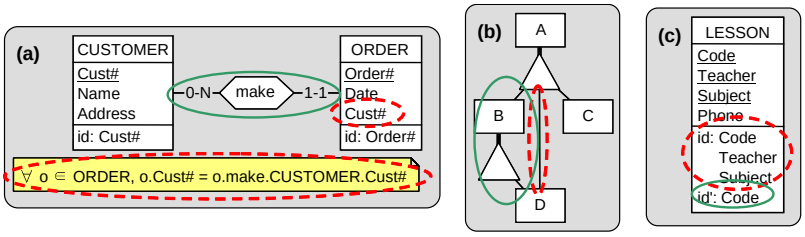

Fig. 4. Examples of redundant constructs.

The second category includes constraints that can be formally inferred from other constructs on the schema. We describe two examples.

- **Transitive is-a relation** (figure 4 *(b)*) Based on the set-theoretic inclusion relation, is-a relations are transitive: if entity type D is a subtype of B, which itself is a subtype of A, then D is also a subtype of A. Specifying the latter property in the schema is needless. Removing it improves simplicity, expressiveness and evolvability.
- **Non-minimal identifiers** (figure 4 *(c)*) Non-minimal identifier I includes components that can be discarded without I loosing its uniqueness property. This pattern can be detected if the minimal subset of I has been declared an identifier of the entity type, as shown in the figure. According to the Armstrong inference rules, the largest identifier can be derived from the smaller one, and therefore can be discarded, which will improve the three quality requirements.

5.3 Foreign Constructs

Foreign constructs are groups of objects that technically comply with the model but that are highly influenced by the modeling practices of another model. Such constructs may appear due to cultural habits of the database designer, to misunderstanding of the *philosophy* (way to perceive the world) of the model or as left-over of a too straightforward migration process. Because of the large variety of models practically used, many different foreign constructs can be found. Let us mention two classical cases:

- **Referencing attribute** Such attribute expresses relationships between concepts of the application domain. The attribute is accompanied by an informal constraint describing its intention. Such pattern is illustrated in figure 4 *(a)*. In order to improve the schema quality, one transforms the attribute into a many-to-one binary relationship type (unless it is, in addition, redundant). This substitution increases the expressiveness of the schema without having a major impact on the other requirements.

– **IMS style** As shown in figure 5 *(a)*, a many-to-many relationship type is expressed in the the legacy IMS style, using three intermediate entity types following the hierarchical database modeling practices [17]. Such construct has obviously a harmful impact on the schema quality. Through semantics-preserving transformations, the source flawed pattern can be replaced by a many-to-many relationship type (figure 5 *(b)*) which significantly increases the three quality requirements.

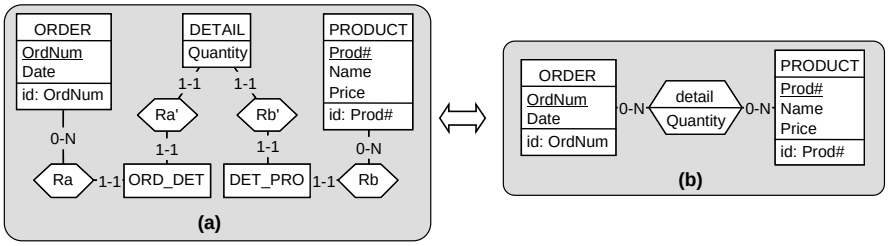

Fig. 5. Example of IMS style construct and its best practice alternative.

5.4 Irregular Constructs

The last category of defects we identified are irregular constructs. They appear in large schemas when similar types of facts are expressed by different types of constructs. This anomaly does not affect individual constructs but the schema as a whole, which appears inconsistent. If these constructs are correct, they belong to the same equivalence class, so that each non-optimal construct can be replaced by a better one from this class. This substitution do not decrease the quality requirements but strengthen the evolvability of the schema.

6 Framework Application

In the previous section, we defined a taxonomy of constructs that can easily be related to the equivalence classes of the framework. This taxonomy suggests a way of using and applying the framework as it gives defect detection criteria and it provides a method for the evaluation and the improvement of schemas. Indeed, the taxonomy (1) brings a well-focused study of defects, (2) is developed in a specific context and (3) provides examples that suggest improvement techniques. In this section, we will discuss the integration of the taxonomy in our framework.

An equivalence class theoretically comprises all the constructs that can be used for representing a specific type of facts, independently of the context. Given a definite context, each construct of a class will receive a score, that defines its level of quality. The taxonomy allows to identify more precisely the constructs

that are considered as defects and therefore to specialize the framework to quality evaluation and improvement in a particular situation.

An important aspect of the framework is the development of ratings, i.e., the definition of quality scales and the application of these scales to the scoring of the constructs. Obviously the definition of ratings for all constructs of each equivalence class, for all possible contexts should be a huge task. Again, the taxonomy can be used to specialize the framework. Simple ratings can be produced using the quality differences between the elements of each couple *(problem, solution)* of the *complex* and the *foreign* construct categories (the *irregular* and *redundant* construct categories will not be used here as they do not provide such evaluations). Example of ratings and a deeper discussion about the evaluation of schema quality using the framework can be found in [1].

The quality difference in a couple is an indicator on the relative scores of these constructs. Obviously a coarse evaluation such as the one provided in the previous section will not allow us to define a fine-grained scale. However, it provides a primitive but usable scale with 2 values *[0,1]*. Through a Condorcet-like voting technique, it is possible to designate the best practice as the construct that has been the most preferred in all the defect fixing suggestions. Then, the 1 score is assigned to the best practice and the 0 score to the other constructs of the class.

Because of the limited information (constructs and quality indicators) available in the taxonomy, the production of more precise ratings should require more investigation. Other scales are discussed in [1], in which we propose for example to use an ordinal scale based on five grades (e.g., *very bad*, *bad*, *neutral*, *good*, *very good*).

Improving the schema following a single quality requirements (e.g., simplicity, expressiveness or evolvability) becomes an easy task. However in practice, quality requirements are often combined and may lead to conflicting suggestions. When combining criteria, two situations appear. In the first one, all the criteria come to the same conclusion, i.e., there exists one best alternative construct that improves all the requirements. In the other situation, there are conflicts between different possibilities and we have to rely on trade-off techniques. For example, we can assign a weight to the requirements and compute an average score if the rating are properly defined (their scale is composed of a sufficient number of values).

7 Conclusion

The principle of taxonomy of defective constructs presented in this paper allows us to refine the quality evaluation and improvement framework proposed in [1], notably since it contributes to populating the equivalence classes. The taxonomy is semi-empirical. It derives from good practices published in the litterature and from modeling experience. The identified defects are probably representative of the common practical defects of this last decade. It also provides designers with guidelines to identify potential problems in database schemas and to ap-

ply solutions according to quality criteria such as *simplicity*, *expressiveness* and *evolvability*. It is important to note that this approach, based on the evaluation of *semantically significant constructs*, does not oppose classical metrics approaches counting atomic objects in the target schema. On the contrary, once defects violating definite quality criteria have been identified, they can be counted and weighted (according to their severity) in order to produce detailed and global metrics.

Though the illustrations (taxonomy and example schemas) of this paper concern the conceptual abstraction level only, the principles we have developed are valid for all abstraction levels and all data modeling paradigms. A demand exists for *relational schema* evaluation, inasmuch as software quality evaluation mainly addresses software metrics at the code level (high level model evaluation still is emerging). At this level, the quality criteria and the taxonomy are specific. For example, *time* and *space performance* as well as *DDL portability* criteria may become important. On the taxonomy side, such defects as *implicit foreign key*, *concatenated columns* and *missing primary key* will appear.

At the present time, we have defined about 20 equivalence classes with their rankings, as well as an extended taxonomy of conceptual defects for Entity-relationship schemas. We are also developing a suite of tools to identify instances of schema patterns (based on a declarative pattern description language), to compute various metrics of these instances and to apply improvement transformations.

The future work will address (1) the validation of the framework [8] and of the tools with the collaboration of experts in database engineering and (2) the extension of the framework to relational database evaluation and improvement.

References

1. Lemaitre, J., Hainaut, J.L.: Transformation-based framework for the evaluation and improvement of database schemas. In Pernici, B., ed.: CAiSE. Volume 6051 of Lecture Notes in Computer Science., Springer (2010) 317–331
2. Hainaut, J.L.: Bases de données: Concepts, utilisation et développement. Dunod (2009)
3. McBrien, P., Poulovassilis, A.: A formal framework for er schema transformation. In: Conceptual Modeling - ER '97, 16th International Conference on Conceptual Modeling, Los Angeles, California, USA, November 3-5, 1997, Proceedings. Volume 1331., Springer (1997) 408–421
4. Codd, E.F.: Normalized data structure: A brief tutorial. In: SIGFIDET Workshop, ACM (1971) 1–17
5. Burton-Jones, A., Weber, R.: Understanding relationships with attributes in entity-relationship diagrams. In: ICIS '99: Proc. of the 20th international conference on Information Systems, Atlanta, GA, USA, Association for Information Systems (1999) 214–228

[8] As suggested by a reviewer, the use of social network could help in the scoring of constructs.

6. Gemino, A., Wand, Y.: Complexity and clarity in conceptual modeling: comparison of mandatory and optional properties. Data Knowl. Eng. **55**(3) (2005) 301–326
7. Assenova, P., Johannesson, P.: Improving quality in conceptual modelling by the use of schema transformations. In: ER '96: Proc. of the 15th International Conference on Conceptual Modeling, London, UK, Springer-Verlag (1996) 277–291
8. Rauh, O., Stickel, E.: Standard transformations for the normalization of er schemata. In: CAiSE, Springer (1995) 313–326
9. Hainaut, J.L.: The transformational approach to database engineering. In Lämmel, R., Saraiva, J., Visser, J., eds.: Generative and Transformational Techniques in Software Engineering. Volume 4143 of LNCS., Springer (2006) 95–143
10. Davis, A., Overmyer, S., Jordan, K., Caruso, J., Dandashi, F., Dinh, A., Kincaid, G., Ledeboer, G., Reynolds, P., Sitaram, P., Ta, A., Theofanos, M.: Identifying and measuring quality in a software requirements specification. In: Proceedings of the First International Software Metrics Symposium. (1993) 141–152
11. Moody, D.L., Shanks, G.G.: Improving the quality of data models: empirical validation of a quality management framework. Inf. Syst. **28**(6) (2003) 619–650
12. ISO/IEC: ISO 9126-1:2001, Software engineering - Product quality, Part 1: Quality model. ISO/IEC (2001)
13. Bansiya, J., Davis, C.G.: A hierarchical model for object-oriented design quality assessment. IEEE Trans. Software Eng. **28**(1) (2002) 4–17
14. ISO/IEC: Software Engineering – Software product Quality Requirements and Evaluation (SQuaRE) – Guide to SQuaRE. ISO/IEC (2005)
15. Si-Said Cherfi, S., Akoka, J., Comyn-Wattiau, I.: Perceived vs. measured quality of conceptual schemas: An experimental comparison. In: ER (Tutorials, Posters, Panels & Industrial Contributions), Australian Computer Society (2007) 185–190
16. Batini, C., Ceri, S., Navathe, S.B.: Conceptual database design: An Entity-relationship approach. Benjamin-Cummings Publishing Co., Inc., Redwood City, CA, USA (1992)
17. Hainaut, J.L.: Hierarchical data model. In: Encyclopedia of Database Systems. (2009) 1294–1300