Consistent Design of Relational Databases using EERCASE

Robson N. Fidalgo, Edson A. Silva

Universidade Federal de Pernambuco, Brazil
{rdnf, eas4}@cin.ufpe.br

Abstract. This article introduces EERCASE, a Computer Aided Software Engineering tool that is based on the best practices of the Model Driven Development paradigm to provide a consistent environment for relational database design. EERCASE follows the graphical notation of the Enhanced Entity–Relationship model according to Elmasri and Navathe, implements the EERMM metamodel to avoid syntactically invalid constructs, shows and describes static semantic errors, and generates data definition code that takes into account advanced structural validations. The theoretical and technical framework used for the implementation of EERCASE is discussed, with emphasis on the restrictive and informative validations performed by it. In addition, considering feedbacks on modeling errors and code generation, EERCASE is also presented as a computational environment that favors active learning.

Categories and Subject Descriptors: D.2 [**Software Engineering**]: Design Tools and Techniques; H.2 [**Database Management**]: Logical Design; H.2 [**Database Management**]: Languages

Keywords: EER, Case tool, SQL

1. INTRODUCTION

There are three levels of database design [Elmasri and Navathe 2016; Silberschatz et al. 2019; Ramakrishnan et al. 2003]: 1) conceptual, which has a high level of abstraction and defines what the database contains; 2) logical, which specifies how to implement the database in a specific data model; and 3) physical, which describes how to implement and optimize the database using a particular technology, addressing its specific aspects and dialects.

In conceptual database design, the Enhanced Entity-Relationship (EER) model is more expressive than the alternative of using a UML class diagram [Bavota et al. 2011]. However, since it was originally proposed [Chen 1976], several other conceptual and graphical reductions and extensions [Song et al. 1995] have emerged. Given this plurality of notations, concepts that exist in one notation may not exist in another or may be interpreted differently. For example: 1) notations without a "discriminator attribute" concept (i.e., partial key) require the use of "identifier attributes" in relationships or weak entities, which is a conceptual mistake; 2) notations that use the same graphical representation for generalization and specialization links require superentities to be drawn above subentities, which is not always possible; 3) notations that do not allow attributes in relationships or N-ary relationships make the design of non-trivial domains less intuitive; and 4) there are notations that read the cardinality and participation of a relationship using only the look-across convention, only the look-here convention, or both of these conventions [Song et al. 1995]. Notations based on the look-across convention for reading participation cannot correctly model this constraint on N-ary relationships, because these notations are unable to represent the mandatory participation of a single entity in the N-ary relationship [Song et al. 1995], as the look-across convention on an N-ary relationship always considers N-1 entities.

Regarding the logical and physical design of databases, because the relational model [Codd 1983] is the most used in recent decades, is still well accepted for conventional-transactional database projects,

and is widely taught in introductory database courses, it is the focus of this study.

The logical design of a relational database, in a simplified way, consists of converting each: 1) entity (i.e., strong or weak) or multivalued attribute into a relation; 2) common, derived, or composed attribute into an attribute of a relation; 3) identifier or discriminator attribute into a Primary Key (PK); 4) 1:1 relationship into a Foreign Key (FK) with the "unique" constraint or into a merged relation; 5) 1:N relationship into a FK that must migrate from the "1" side to the "N" side and must not have a "unique" constraint; 6) M:N relationship into a relation whose PK is formed by FKs for involved relations; 7) N-ary relationship into a relation whose PK is composite (the number of fields depends on the degree and cardinality of the relationship, as well as whether it has a discriminator attribute) and has one FK for each relation involved; 8) associative entity into a relation whose PK is normally formed by the FKs for the relations involved (the number of fields that form the PK also depends on the degree and cardinality of the relationship, as well as whether it has a discriminator attribute); and 9)inheritance in a single (i.e., merged) relation or a set of relations whose PKs can also be FKs. The total participation of an entity in a relationship can be guaranteed by the "not null" constraint or through the implementation of triggers, which will be discussed in Section 2.2.

Details on logical design can be found in [Elmasri and Navathe 2016; Silberschatz et al. 2019; Ramakrishnan et al. 2003]. In turn, the physical design of a relational database can be summarized as a rewriting of the logical schema so that it: conforms to the Data Definition Language (DDL) syntax of the chosen Relational Database Management System (RDBMS), contains the indexes to optimize data access, and includes triggers or assertions to ensure advanced structural constraints (e.g., disjoint and complete inheritances).

Although there are many Computer Aided Software Engineering (CASE) tools that can assist in the conceptual, logical, and physical design of relational databases, no CASE tool, other than EERCASE[1], was found in the literature that supports Elmasri and Navathe notation [Elmasri and Navathe 2016] and offers consistent support for database design, especially regarding the implementation of advanced structural validations (cf. Section 2.2). In this context, the goal of this article is to present how EERCASE helps: 1) prevent syntactic errors (i.e., grammatically invalid constructions); 2) report and describe static semantic errors (i.e., grammatically valid constructions that have some structural contradiction); and 3) generate Structured Query Language/Data Definition Language (SQL/DDL) code with rules that ensure advanced participation constraints in relationships (e.g., 1:N relationships with total participation on 1 side - cf. Figure 2) and complete and disjoint inheritances (cf. Figures 7 and 8). This article extends [Silva and Fidalgo 2021] by detailing the validations provided by EERCASE, describing the constructors of its metamodel (EERMM), and explaining how the conceptual, logical, and physical designs of a relational database are performed using EERCASE.

Regarding related studies, brModelo[2] and TerraER[3] are important tools for database design, however, the notation used in brModelo considers the look-across convention for reading the participation of an entity in the relationship and does not have the concept of a discriminator attribute, while TerraER does not generate SQL/DDL code. Furthermore, these tools do not adhere to the static semantic rules proposed by Dullea, Song, and Lamprou [Dullea et al. 2003] and by Calvanese and Lenzerini [Calvanese and Lenzerini 1994]. For example, brModelo and TerraER are not able to capture the static semantic errors presented in Figure 12-B. Other important tools, such as DBDesigner, Erwin, and ER/Studio, do not allow modeling attributes in relationships or N-ary relationships, as these tools do not have a diamond constructor to represent a relationship. It is important to highlight that these tools are not based on the Model Driven Development paradigm. That is, unlike EERCASE, they are not based on a metamodel, nor take advantage of the facilities provided by a specific framework (e.g., epsilon - cf. Section 2) to create CASE tools.

[1]http://www.sites.google.com/a/cin.ufpe.br/eercase
[2]http://www.sis4.com/brModelo
[3]http://www.terraer.com.br

The remaining sections of this article are organized as follows. Section 2 presents the theoretical-technological framework used to develop EERCASE and discusses the validations made by this tool. Section 3 describes how to use the tool and its principal features. Section 4 talks about the educational potential of EERCASE. Finally, Section 5 presents final considerations and possibilities for future studies.

2. EERCASE DEVELOPMENT

The theoretical framework used to specify EERCASE is based on the Model-Driven Development (MDD) paradigm [Brambilla et al. 2017]. In this framework, a modeling language can be specified from a concrete syntax (i.e., graphical notation), an abstract syntax (i.e., metamodel or diagrammatic grammar), static semantics (i.e., rules for good structural formation), and algorithms that transform diagrams into executable code. The technological framework used to build the EERCASE tool was the Graphical Modeling Project (GMP)[4] (used to build CASE tools in Java/Eclipse) and the Epsilon Framework[5] (used to simplify the creation of CASE tools in GMP). Furthermore, EERCASE uses the XML Metadata Interchange (XMI) to store, manipulate, retrieve, and exchange metadata, and works as a standalone application using the Rich Client Platform (RCP) architecture of the Eclipse framework, allowing the tool to work independently of platform and be easy to distribute.

The EERCASE development process begins with the coding of the Enhanced Entity Relationship Metamodel (EERMM) (cf. Section 2.1) in the Emfatic language of the Epsilon framework. This code, in addition to the part corresponding to the EERMM metamodel (i.e., the abstract syntax of EERCASE), also contains the graphical notation defined by Elmasri and Navathe (i.e., the concrete syntax of EERCASE). Then, the validations ensured by the static semantic rules are coded using the Epsilon Validation Language (EVL), and the EER diagrams are transformed into an SQL/DDL (PostgreSQL) script with triggers to ensure advanced constraints (e.g., complete or disjointed inheritances - cf. section 2.2), using the Epsilon Transformation Language (ETL). Finally, the EuGENia tool, also from the Epsilon Framework, is used to implement the configuration and execution details specified by GMP. In order to facilitate and accelerate the generation of scripts to build the database, it was decided that EERCASE could abstract the transformations from the conceptual schema to the logical schema, and from there to the physical. For this process, when an EER construct has more than one possible manner in which it can be mapped (e.g., 1:1 relationships and inheritances), the most general manner is used.

Considering the context presented above, the EERCASE transformations are predefined as follows: 1) each strong, weak, generalized, specialized, or category entity is mapped to a relation/table; 2) each simple attribute (i.e., common, identifier, discriminator, or derived) is mapped to an attribute/field in its respective relation/table; 3) each composite attribute is mapped to a set of attributes/fields in its respective relation/table; 4) each multivalued attribute is mapped to a new relation/table with a FK for the base relation/table; 5) each 1:N relationship is mapped to a FK on the "N" side; 6) each 1:1 relationship is mapped to a FK on the "total" side (if the participation is total-total or partial-partial, the designer must choose the FK - cf. Figure 10; 7) each M:N, N-ary, or Associative Entity relationship is mapped to a new relation/table whose PK and FKs depend on the degree and cardinality of the relationships, as well as whether they have discriminator attributes; 8) each inheritance relationship is mapped to a FK (which is also a PK) from the specialized to the generalized relation/table; and 9) each category relationship is mapped to a FK from each relation/table that maps a specialized category to the relation/table that maps the generalized category. Participation, Disjointness, and Completeness constraints are discussed in Section 2.2. The following section introduces the EERMM metamodel [Fidalgo et al. 2012; Fidalgo et al. 2013].

[4]http://www.eclipse.org/modeling/gmp
[5]http://www.eclipse.org/epsilon/doc/book

2.1 EERMM Metamodel

A metamodel describes the abstract syntax (i.e., grammar) of a modeling language. In other words, a metamodel defines the concepts of a modeling language, its properties, and the valid associations between its concepts. That is, a metamodel specifies the rules that make it possible to distinguish between valid and invalid diagrams. In this sense, a metamodel is as useful to a modeling language as a grammar is to a programming language. Figure 1 shows the EERMM Metamodel [Fidalgo et al. 2012; Fidalgo et al. 2013] used in the development of the EERCASE tool.

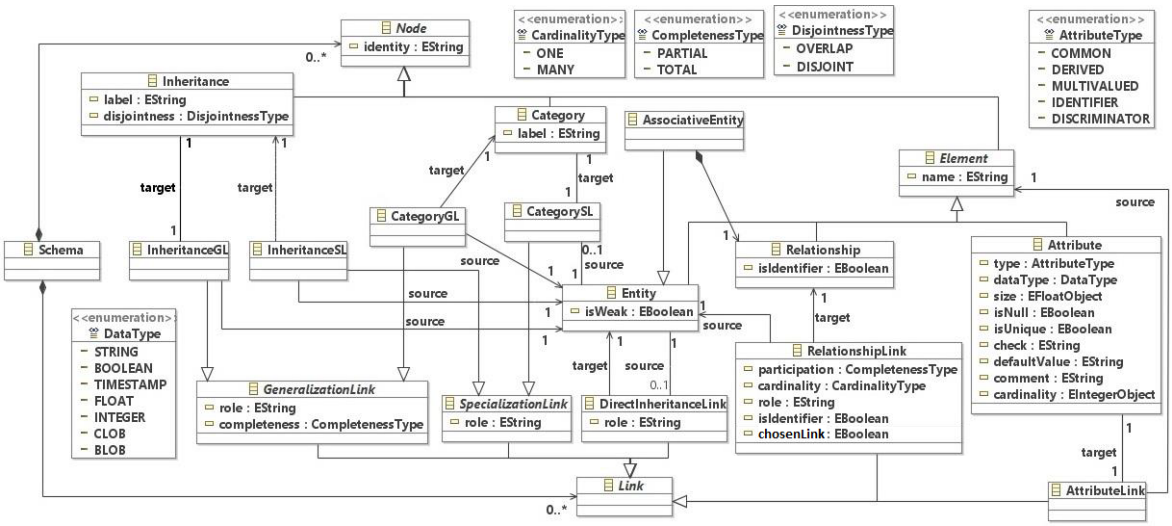


Fig. 1. The EERMM metamodel

In Figure 1, the Schema metaclass is composed of the Node and Link metaclasses. The Node metaclass is specialized in the Inheritance, Category, and Element metaclasses, which is specialized in the Attribute, Relationship, and Entity metaclasses, with the latter being specialized in the AssociativeEntity metaclass. Link metaclasses have a Node as a source and another Node as a target. Furthermore, the Link metaclass is specialized in the following metaclasses: AttributeLink, RelationshipLink, DirectInheritanceLink, and GeneralizationLink, which is specialized in InheritanceGeneralizationLink (InheritanceGL) and CategoryGeneralizationLink (CategoryGL). Finally, SpecializationLink is specialized in InheritanceSpecializationLink (InheritanceSL) and CategorySpecializationLink (CategorySL). A more detailed explanation of the EERMM is beyond the scope of this work. However, the complete specification of the EERMM can be found in [Fidalgo et al. 2012; Fidalgo et al. 2013].

2.2 EERCASE Validations

The validations performed by EERCASE can be either restrictive or informative. The following paragraphs explain restrictive validations, followed by a description of informative validations.

Restrictive validations prevent the occurrence of syntactic errors or violation of Participation, Disjointness and Completeness. Syntactic validations are guaranteed in the conceptual design through the implementation of the EERMM metamodel. Table I presents the syntactic validations supported by EERMM. In turn, the other validations are guaranteed in the logical and physical designs by defining mandatory FKs (i.e., "not null") or by implementing triggers, respectively.

Validations using mandatory FKs are considered basic, because they are assured only by the "not

Table I. Syntactic validations supported by EERMM

#	Rule
1	A Node cannot be connected to another Node without a Link, and vice-versa. That is, a node cannot be directly connected to other nodes nor can a link be directly connected to other links.
2	A Link cannot be created without a Node as a source and another as a target. That is, an isolated Link cannot be depicted in the drawing area.
3	A RelationshipLink cannot connect an Entity to others nor a Relationship to others. That is, a RelationshipLink cannot exist between entities nor between relationships.
4	An Inheritance or a Category cannot have an Attribute and an Attribute cannot be linked to more than one Element. That is, an Attribute only belongs to an Entity, an AssociativeEntity, a Relationship, or another Attribute (composite attribute).
5	An Inheritance can be disjoint or overlap, total or partial, and have many SpecializationLinks, but cannot have more than one GeneralizationLink. In other words, an Inheritance can be specialized in many subentities, but must be generalized to only one superentity.
6	A Category can have neither overlapping multiple SpecializationLinks, but it can have many GeneralizationLinks. That is, a Category is disjoint, having only one subentity, but potentially many superentities.
7	An Entity cannot be source of more than one CategorySL. That is, a subentity can have only one CategorySL.
8	An Entity cannot be source of more than one DirectInheritanceLink. That is, a superentity can have only one DirectInheritanceLink.

null" constraint. For example 1:N relationships with total participation on the "N" side or 1:1 relationships with total participation on the side that will receive the FK (cf. Figure 2).

In Figure 2, using the "look-here" convention for participation and the "look-across" convention for cardinality, it can be seen that every employee must work for a department and that every department must have at least one employee working on it. Furthermore, every department must have an employee as a manager, but not every employee must manage a department. Note that the total participations, highlighted in red, are easily guaranteed using mandatory FKs ("not null") and are therefore considered basic. In addition to "not null", the FK that implements the "Manages" relationship must also have the "Unique" constraint.

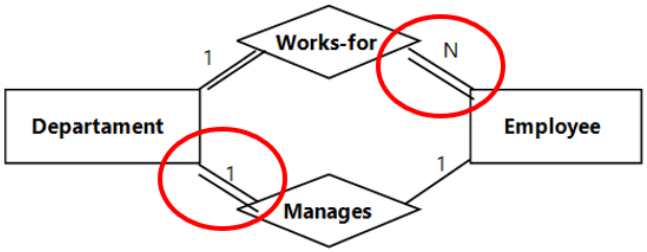


Fig. 2. Example of basic structural constraint

Validations that require triggers to be enforced are considered advanced, for example: 1) 1:N relationships with total participation of "1" side - cf. Figure 3; 2) 1:1 relationships with total-total participation - cf. Figure 4; 3) N:N relationships with total participation on any side (or both) - cf. Figure 5; 4) N-ary relationships with total participation on any side (including all and regardless of relationship cardinality) - cf. Figure 6), and 5) complete or disjoint inheritances - cf. Figure 7 and Figure 8. In summary, considering the transformations predefined by EERCASE (cf. Section 2), participation and completeness constraints require triggers to ensure that every PK value is referenced

by at least one FK value. These triggers must ensure the following two situations: when including or updating a PK value, there must always be a corresponding FK value, and when excluding or updating an FK value, there will be no PK value without a corresponding FK value. In the case of the Disjointness constraint, another trigger is needed to ensure that the PK value of a generalized entity is referenced by only one FK value of among its specialized entities. The figures below show examples for these types of validations.

In Figure 3, the total participation highlighted in red requires two triggers. The first ensures that, when inserting a department or updating its PK value, there must be at least one employee working in it. The second trigger ensures that, when deleting an employee or updating its department FK, there will be no department without an employee.

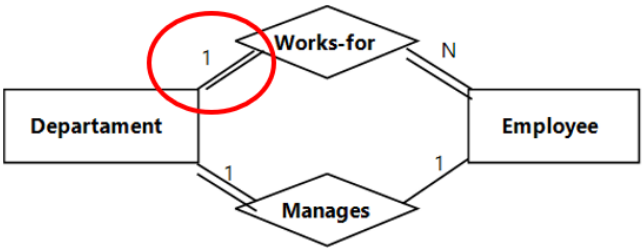


Fig. 3. Example of advanced structural constraint on an 1:N relationship with total participation on side 1

In Figure 4, assuming that the FK is in the department, the total participation highlighted in red will also require two triggers. The first is to enforce that, when inserting a budget or updating its PK value, there will be a department related to it. The second is to prevent deleting a department without also deleting its budget, as well as to avoiding updating the FK for the budget without also updating the PK value of budget. Note that the total participation highlighted in green is basic as it can be easily enforced by setting the FK with both "unique" and "non-null" constraints.

Fig. 4. Example of advanced structural constraint on an 1:1 relationship with total-total participation

In Figure 5, similar to the previous two examples, the total participation highlighted in red requires a trigger to ensure that, when inserting a project or updating its PK value, there must be at least one employee associated with it, plus another trigger to ensure that, when deleting an employee or updating its FK from "Works-on" to "Project," no project will be without an employee.

Fig. 5. Example of advanced structural constraint on an N:N relationship with total participation on either side

In Figure 6, following the same reasoning, the total participation highlighted in red requires a trigger to enforce that, when inserting equipment or updating its PK value, there must be at least one employee and one project related to it. The other trigger is to assure that, when deleting an employee

or a project, as well as when updating the PK/FK of a equipment in the "Handle" relationship, there will be no equipment lacking links to an employee and to a project.

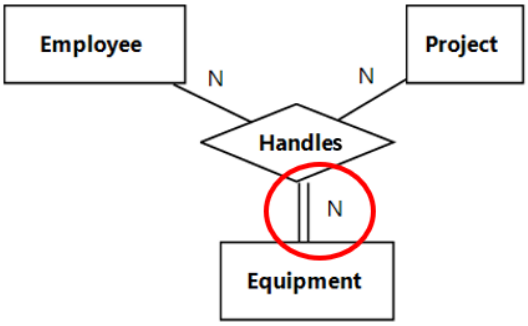


Fig. 6. Example of advanced structural constraint on an N-ary relationship with total participation on either side

In Figure 7, the total completeness highlighted in red defines that every project is specialized in public and/or private. The implementation of this constraint also requires two triggers. The first trigger ensures that, when inserting a generalized project or updating its PK value, there will be a specialized project linked to it. The second ensures that, when deleting a specialized project or updating its PK/FK, there will be no generalized project that is not specialized.

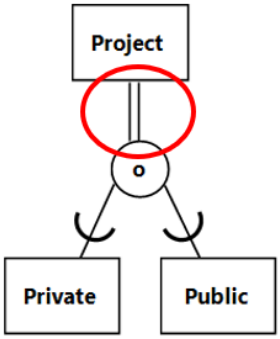


Fig. 7. Advanced structural constraint example in total inheritance

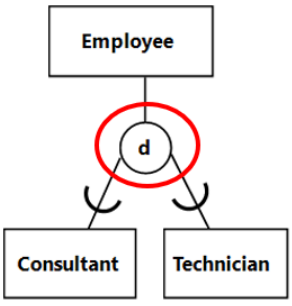


Fig. 8. Advanced structural constraint example in disjoint inheritance

In Figure 8, the red highlighted disjunction, that prevents an employee from being a consultant and technician at the same time, also requires two triggers. The first ensures that, when inserting a generalized employee or updating its PK value, there will only be one specialized employee linked

to it. The second ensures that, when updating the PK/FK of a specialized employee, no employee should be a consultant and technician at the same time.

Informative validations are static semantic rules that report constructions that are syntactically valid according to the EERMM, but which have conceptual errors or structural contradictions according to Dullea, Song and Lamprou [Dullea et al. 2003] or Calvanese and Lenzerini [Calvanese and Lenzerini 1994]. Table II shows some conceptual errors and Figure 9 illustrates examples of constructions that have structural contradictions. In Figure 9 The first four schemes illustrate the structural contradictions stated in the Corollaries presented by Dullea, Song and Lamprou [Dullea et al. 2003], namely: (a) Corollary 1 - "All 1:1 recursive relationships with mandatory–optional or optional–mandatory minimum cardinality constraints are structurally invalid"; (b) Corollary 2 - "All 1 : M or M : 1 recursive relationships with mandatory–mandatory minimum cardinality constraints are structurally invalid"; (c) Corollary 3 - "All 1 : M or M : 1 recursive relationships with mandatory participation constraint on the 'one' side and an optional participation constraint on the 'many' constraints are structurally invalid", and (d) Corollary 4 - "Cyclic paths containing no opposing relationships and no self-adjusting relationships are structurally invalid and called a Circular Relationship". In turn, the other three schemes (e, f, and g) have the same structural contradiction defined by Calvanese and Lenzerini [Calvanese and Lenzerini 1994], which presented an formal method for reasoning about a set of inheritance and cardinality constraints. According to their formalization, a superentity with a mandatory minimum cardinality in a binary relationship with one of its subentities (i.e., each superentity instance must be related with at least one subentity instance) forces the number of subentity instances to be greater than the number of superentity instances, which is a contradiction.

Table II. Examples of conceptual contradictions reported by EERCASE

Concept	Error
Strong Entity	1. Lack of identifier attribute; 2. Use of discriminator attribute or 3. Presence of attributes with repeated names.
Relationship	1. Existence of identifier attribute or 2. Use of discriminator attribute when the cardinality is 1:N or 1:1.
Identifying Relationship	1. Absence of strong entity; 2. Definition of partial participation on the weak entity side; 3. Use of identifier attribute on weak entity; 4. Use of "N" cardinality on the strong entity side; 5. Definition of weak entity with no discriminator attribute when the cardinality is 1:N; 6. Definition of weak entity with discriminator attribute when the cardinality is 1:1 or 7. Cyclic construction of identifying relationships.
Associative Entity	1. Presence of identifier attribute or 2. Use of discriminator attribute when the cardinality is 1:N or 1:1.
Inheritance	1. Definition of a specialized entity with an identifier or discriminator attribute; 2. Definition of a specialized entity as a weak entity or 3. Cyclic construction of inheritance.

3. USE OF EERCASE

Figure 10 shows the EERCASE graphical interface. In this figure, area "A" shows the constructors in Elmasri and Navathe notation, area "B" delimits the drawing region of the diagrams and, finally, area "C" presents the properties that can be configured for each EER constructor selected in the diagram. The design of an EER diagram begins by clicking on the desired constructor (area "A") and then click

where it should appear in the drawing area (area “B”). Following this, the properties of a selected constructor (area ”C”) can be edited and new constructors can be added to the diagram. In order to demonstrate the expressiveness of EERCASE, in area ”B” of Figure 10, a basic EER schema is modeled (due to space limitations, only necessary attributes are shown).

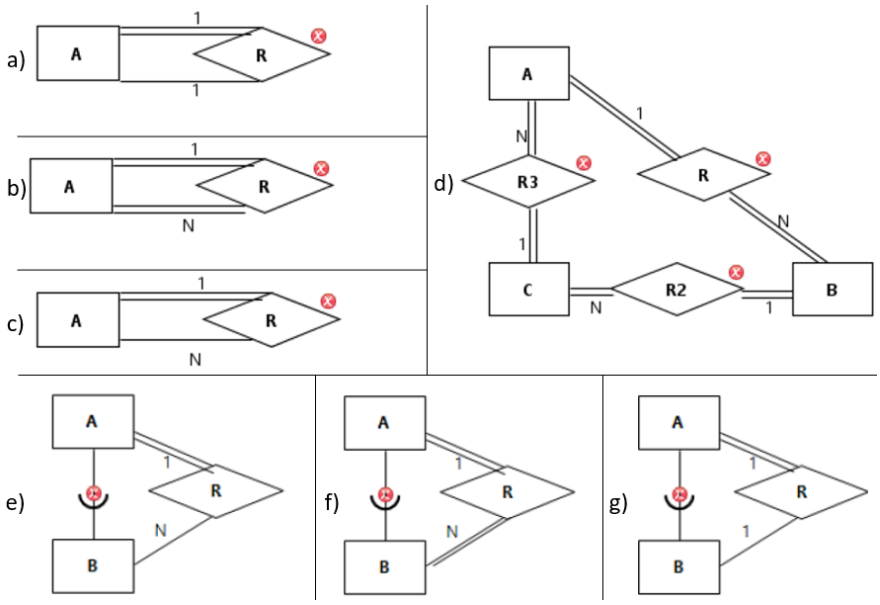


Fig. 9. Examples of structural contradictions reported by EERCASE [Dullea et al. 2003; Calvanese and Lenzerini 1994]

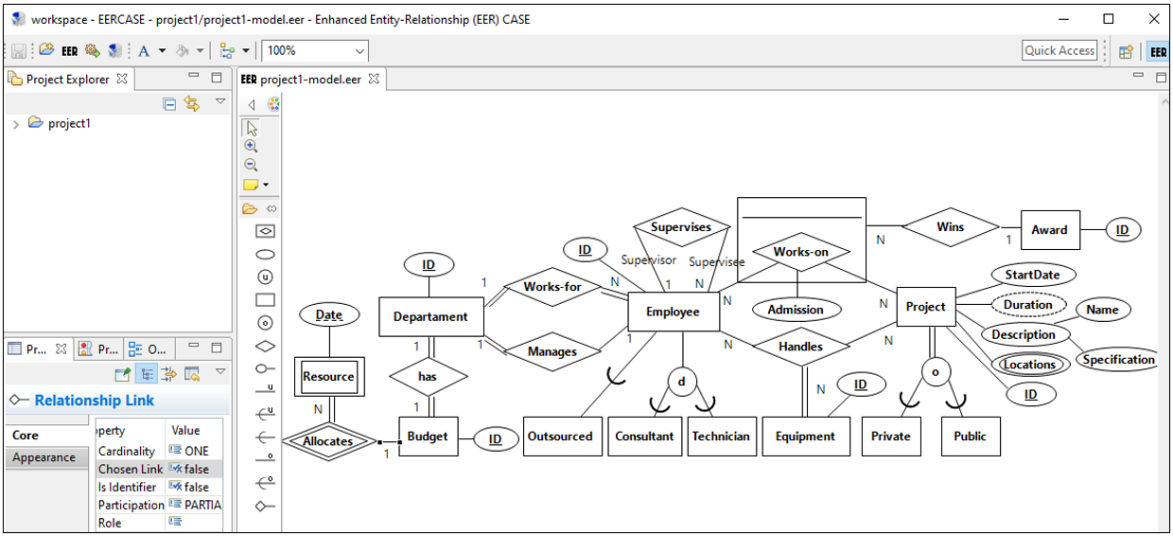


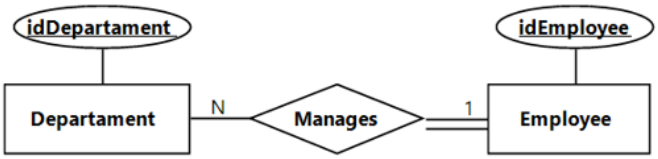
Fig. 10. EERCASE graphical interface

As can be seen in Figure 10-B, EERCASE supports the modeling of the EER constructors according to Elmasri and Navathe [Elmasri and Navathe 2016], namely: 1) Regular Entity (all except Resource); 2) Associative Entity (Works-on); 3) Weak Entity/Identifying Relationship (Resource/Allocates); 4) Simple Attribute (StartDate); 5) Composite Attribute (Description); 6) Multivalued Attribute (Locations); 7) Derived Attribute (Duration); 8) Identifier Attribute (ID); 9) Discriminator Attribute

(Date); 10) Attribute on Relationship (Admission); 11) Self-relationship (supervises); 12) N-ary relationship (Handles); 13) Participation Constraint (double or single line); 14) Cardinality Constraint ("1" or "N"); 15) Roles on Relationship (Supervisor and Supervisee); 16) Direct Inheritance (a semi-circle point to generalized entity); 17) Disjoint or Overlapping Inheritance (a circle with the letter "d" or "o", respectively); and 18) Complete or Partial Inheritance (a double or a single line linked to a circle, respectively).

Because EERCASE is developed on the Eclipse platform, it also supports features such as: creation of text notes, definition of colors for the constructors, viewing of various schemas in different tabs, control of zoom, and the ability of undo or redo an action. EERCASE is freely distributed under the Creative Commons license with attribution without derivations (CC BY-ND 2.0 BR) for Windows and Linux platforms. More information, including a download link, demo videos, documentation and publications, can be found at http://sites.google.com/a/cin.ufpe.br/eercase.

To give an example of advanced constraint code generation in EERCASE, Figure 11-A shows a schema with a 1:N relationship having total participation on the "1" side, while Figure 11-B shows the PostgreSQL trigger generated to in order to enforce total participation on the "1" side.



A) EER schema illustrating non-trivial structural constraints

```
CREATE FUNCTION FUNCTION_TRG_Manages_Departament()
RETURNS TRIGGER AS $BODY$
    DECLARE countEmployee INTEGER;
    BEGIN
        SELECT COUNT(*) INTO countEmployee
        FROM Departament
        WHERE Departament.idEmployee = OLD.idEmployee;

        IF countEmployee = 0 THEN
            RAISE EXCEPTION 'TRG_Manages_Departament';
        END IF;

        IF (TG_OP = 'DELETE') THEN
            RETURN OLD;
        ELSIF (TG_OP = 'UPDATE') THEN
            RETURN NEW;
        END IF;

    END
$BODY$
LANGUAGE plpgsql VOLATILE;

CREATE CONSTRAINT TRIGGER TRG_Manages_Departament AFTER
DELETE OR UPDATE OF Manages_Employee_idEmployee ON
Departament INITIALLY DEFERRED
FOR EACH ROW EXECUTE PROCEDURE
FUNCTION_TRG_Manages_Departament();
```

```
CREATE FUNCTION FUNCTION_TRG_Manages_Employee()
RETURNS TRIGGER AS $BODY$
    DECLARE countEmployee INTEGER;
    BEGIN
        SELECT COUNT(*) INTO countEmployee
        FROM Departament
        WHERE idEmployee = NEW.idEmployee;

        IF countEmployee = 0 THEN
            RAISE EXCEPTION 'TRG_Manages_Employee';
        END IF;

        RETURN NEW;
    END
$BODY$
LANGUAGE plpgsql VOLATILE;

CREATE CONSTRAINT TRIGGER TRG_Manages_Employee AFTER
INSERT ON Employee INITIALLY DEFERRED
FOR EACH ROW EXECUTE PROCEDURE
FUNCTION_TRG_Manages_Employee();
```

B) Trigger (PostgreSQL) to implement mandatory participation on side 1

Fig. 11. Modeling and code generation of non-trivial constraints in EERCASE

4. EERCASE AS AN EDUCATIONAL TOOL

CASE tools are used in a number of areas of knowledge. In education, based on feedback about modeling errors in a given domain and on an essentially practical interaction, these tools help consolidate

student learning, as they set up scenarios that simulate real-life situations. In other words, CASE tools provide a computational environment that favors active learning, with the advantage of being able to give individualized feedback and generate standardized code that is free of syntactic and static semantic errors, useful resources that help the content learned be retained for a longer time and that motivate the search for additional knowledge.

Since 2012, EERCASE has been recommended for use in the Centro de Informática at UFPE as part of the basic database course and, during the COVID-19 pandemic period, EERCASE has been a useful tool to support remote educational activities. Through its graphical feedbacks, which indicate whether constructions are prohibited (cf. Figure 12-A) and point out where the errors are (cf. Figure 12-B), as well as through together with explanations of the errors (cf. Figure 12-C), EERCASE has supported learners and answered their questions, helping them to better deal with the particularities of emergency remote classes.

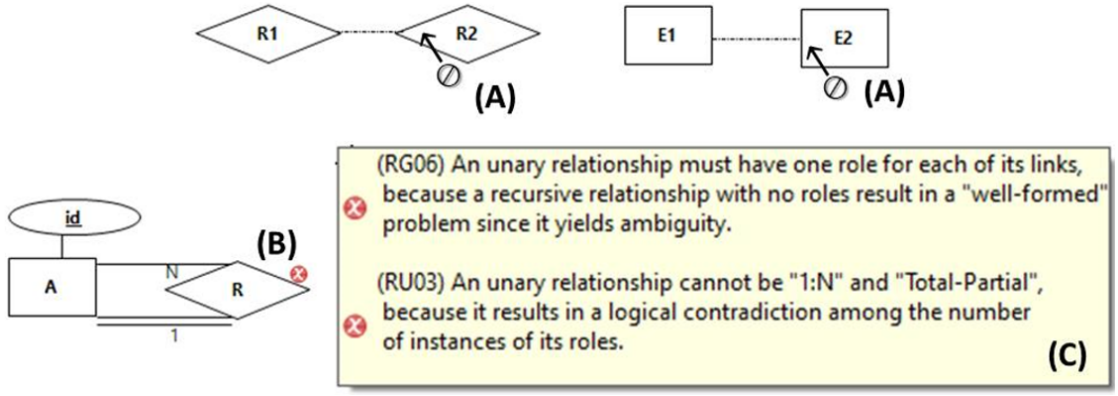


Fig. 12. Graphical feedbacks displayed by EERCASE

5. FINAL CONSIDERATIONS

CASE tools can provide feedback on syntactic and static semantic errors, as well as generating executable code that takes into consideration the implementation of advanced constraints. Feedback are important resource to teach good database conceptual modeling practices, because this feedback provides information on what errors are occurring and where they are. In turn, the generation of code with advanced rules helps in learning complex concepts, such as triggers. CASE tools with these resources therefore favor the learning process, helping to consolidate acquired knowledge and stimulating the search for new knowledge. This article presents EERCASE as a consistent tool for conceptual database design. EERCASE is based on EERMM metamodel, is built on the MDD paradigm and the Epsilon Framework, and is freely distributed (CC BY-ND 2.0 BR) for Windows and Linux. EERCASE abstracts transformations to map from the conceptual schema to the logical schema, and from there to the physical. On the one hand, this abstraction aims to facilitate and accelerate the generation of scripts for building the database. On the other hand, this abstraction limits the database designer to performing a custom transformation. As a potential future study, there is a proposal for a collaborative web-based version.

REFERENCES

Bavota, G., Gravino, C., Oliveto, R., Lucia, A. D., Tortora, G., Genero, M., and Cruz-Lemus, J. A. Identifying the weaknesses of uml class diagrams during data model comprehension. In *International Conference on Model Driven Engineering Languages and Systems*. Springer, Berlin, Heidelberg, pp. 168–182, 2011.

Brambilla, M., Cabot, J., and Wimmer, M. Model-Driven Software Engineering in Practice: Second Edition. *Synthesis Lectures on Software Engineering* 3 (1): 1–207, 2017.

Calvanese, D. and Lenzerini, M. On the interaction between isa and cardinality constraints. In *Proceedings of 1994 IEEE 10th International Conference on Data Engineering*. IEEE, Houston, TX, USA, pp. 204–213, 1994.

Chen, P. P.-S. The entity-relationship model—toward a unified view of data. *ACM transactions on database systems (TODS)* 1 (1): 9–36, 1976.

Codd, E. F. A relational model of data for large shared data banks. *Communications of the ACM* 26 (1): 64–69, 1983.

Dullea, J., Song, I.-Y., and Lamprou, I. An analysis of structural validity in entity-relationship modeling. *Data & Knowledge Engineering* 47 (2): 167–205, 2003.

Elmasri, R. and Navathe, S. B. *Fundamentals of Database Systems, Seventh Edition*. Person, Boston, MA, USA, 2016.

Fidalgo, R., Alves, E., España, S., Castro, J., and Pastor, O. Metamodeling the enhanced entity-relationship model. *JIDM* 4 (3): 406–420, 2013.

Fidalgo, R. D., Souza, E. M. D., España, S., Castro, J. B. D., and Pastor, O. Eermm: a metamodel for the enhanced entity-relationship model. In *International Conference on Conceptual Modeling*. Springer, Berlin, Heidelberg, pp. 515–524, 2012.

Ramakrishnan, R., Gehrke, J., and Gehrke, J. *Database management systems*. Vol. 3. McGraw-Hill, New York, 2003.

Silberschatz, A., Sudarshan, S., and Korth, H. F. *Database System Concepts*. Vol. 7. McGraw-Hill, New York, 2019.

Silva, E. and Fidalgo, R. Eercase: Uma ferramenta robusta para projeto conceitual de banco de dados. In *Anais Estendidos do XXXVI SBBD*. SBC, Porto Alegre, RS, Brasil, pp. 87–92, 2021.

Song, I.-Y., Evans, M., and Park, E. K. A comparative analysis of entity-relationship diagrams. *Journal of Computer and Software Engineering* 3 (4): 427–459, 1995.