Transformation of Entity Relationship Models to Relational Models - A Practical Approach

Gregor Grambow
Aalen University
Aalen, Germany
Email: gregor.grambow@hs-aalen.de

Simon Ruttmann
eXXcellent solutions GmbH
Ulm, Germany
Email: simon.ruttmann@exxcellent.de

***Abstract*—Both Entity Relationship (ER) modeling and the relational model have come a long way and are both prevalent. The former for creating conceptual database schemata and the latter for creating technical database schemata. Unfortunately, both of these models are not directly compatible. Over the decades, various theoretic approaches for transforming ER models to relational models have been proposed. These efforts, however, did not yield practical approaches and thus editors capable of such transformations. Modern editors either have no transformation facilities, or do not use the ER model proposed by Peter Chen but rather provide somehow enhanced database diagrams. Therefore, users have to manually transform ER models to relational models, which is cumbersome and error-prone. To tackle this problem, we propose an automatic transformation from ER models to relational ones that focuses on practical applicability and operational semantics. In particular, the approach enables transformations of the original ER model by Chen as well as prevalent extensions. We have proven the applicability of this approach in two ways: First, we have created a graphical editor capable of flexibly modeling ER diagrams and automatically transforming them to relational models. Second, we have conducted a practical study with our approach and the created editor. Our findings show, that the transformation approach works correctly and the implemented editor makes it accessible to users.**

***Index Terms*—Entity Relationship; ER Model; Relational Model; Database; Editor**

I. Introduction

This article is an extension to the conference paper "A Practical Automated Transformation of Entity Relationship Models to Relational Models" published in 2023 [1]. Entity Relationship (ER) modeling is a prevalent option for semantic data modeling primarily applied to database schemata. This way of modeling has been used since over 40 years. It was introduced in 1976 by Peter Chen [2] and has been the focus of active research for decades. There has been a myriad of extensions to the model, like the ECR model [3], the ECR+ model [4], HERM [5], or the EER model [6]. Many of these extensions offer valuable additions to the basic ER models. Some features have been adopted but many have also been discarded. As a result of this, basic ER modeling became prevalent but with a high number of different flavors. The usage of generalization concepts, cardinality constraints or the application of the n-ary relationships as proposed by Peter Chen differs in many applications.

However, to be usable in a relational database, ER models have to be converted to relational models. Manually executed, this process can be tedious and error-prone. Thus, various approaches for standardized transformations have been proposed, like [3] [7] - [13]. Most of these approaches have two major downsides: First, they often impose certain constraints on the ER models and second, they remain rather theoretic. In many cases, after presenting their approach, the authors recommended them to be used by practitioners or in automated tools. Despite having a clear formalism, a practical implementation was never achieved, often due to the lack of operational semantics. On the other hand, there is a high number of editor tools for ER models, contained in drawing tools [14] - [17], in client software of databases [18] [19], low-code plattforms [20] [21], or in modeling tools like Enterprise Architect [22].

These editors have two main issues: Each of them uses a different subset of ER concepts, some are even closer to a relational or even UML class diagram editor. In addition, the transformation aspect has been ignored almost completely. Thus, semantic data modeling for databases usually involves first drawing an ER diagram and then manually transforming it to database tables, which can be a source of numerous issues.

Contrary to the above mentioned tools the proposed approach stands out for its depth and flexibility when it comes to dealing with ER diagrams. It can capture complex relationships between entities, relations and a variety of attribute types. This enables a more accurate representation of the underlying data structure, resulting in better maintainable and more efficient databases.

To counteract the mentioned transformation issues, we propose an approach for automatically transforming ER models to relational models. As opposed to prior approaches we do not focus on mathematical definitions or calculus but rather on practical applicability and operational semantics. Thereby, our approach can be easily applied to editors in different programming languages. To prove this applicability we have implemented a graphical ER editor that is capable of this automatic transformation.

The most significant aspect of the proposed algorithms and presented tool is thus removing the gap between real ER models and relational models. The proposed algorithms enable modelers to perform work on both the conceptual and technical level. As business requirements can be clearly articulated on a purely conceptual level the proposed approach provides a way for better understanding and representing the data's

semantic structure. This contributes to more precise and better comprehensible relational models.

There are two main extension points of this article to the conference paper "A Practical Automated Transformation of Entity Relationship Models to Relational Models" published in 2023 [1]. Firstly, within the practical implementation section, the validation mechanisms performed during the er modeling process, which are developed and integrated into the modeling tool are also described extensively. The second point of extension is the evaluation of the applicability of the developed modeling tool. The evaluation focuses on comparing the ER modeling process experience utilizing the presented tool against other prevalent tools.

The rest of this paper is organized as follows: Section II discusses related approaches, while Section III defines the concrete style of ER diagram that is assumed as basis for the transformation. Section IV provides an extensive description of the transformation approach. After that, Section VI shows a practical evaluation of the proposed approach followed by Section VII providing a conclusion as well as future directions.

II. Related Work

In this section, we cover two types of related work: Scientific approaches presented for transforming ER models into relational models and practical ER editor tools.

In the scientific community, the case of transforming ER models has been extensively discussed over the decades. Most proposed approaches date back to the 1980s and 1990s. As mentioned, one big issue concerning general applicability is the high number of different ER variants. The basic variant proposed by Peter Chen [2] includes n-ary relationships and weak entities but no generalization concepts. Most of the extensions [3] - [6] to that model focused on adding structures for generalization. Due to that, many different transformation approaches take different variants of the ER model as basis. Most attention was paid to the original version of the ER model [7] - [10] and extended ER models with generalization structures [3] [11] - [13]. All of them have in common that they remain rather theoretic and do not consider operational issues [23]. Further, to the best of our knowledge, none of these approaches lead to the development of a prevalent ER editor tool.

The fact that most research regarding that topic was carried out decades ago lets us assume, that the transformation approaches have been adopted and are now prevalent in ER editors. Therefore, the second part of this section deals with contemporary ER editors. However, before investigating the transformation capabilities, another issue has to be dealt with: There is a high number of ER editors that do not use the ER model as proposed by Peter Chen. Many of them focus on simplified binary relationships. In addition, they do not cover the most prevalent extension to Chen's model: generalization. Such diagrams are often nearer to a relational database diagram than to a real ER diagram. One category featuring such diagrams is drawing tools like Lucid Chart [14], Draw.io [15], or Visual Paradigm [16]. Some of them even contain concepts like stored procedures or triggers and can thus not be considered ER editors. Furthermore, none of them provides a transformation approach. Another category providing such diagrams is database client software, e.g., from PostgreSQL [18] or MySQL [19]. In contrast to the database-specific editor tools there are also low-code platforms such as Mendix [21] and Outsystems [20] to mention. These tools also contain editors to make it possible to model relational models. The above listed tools provide visual modeling that can be directly used as database tables.

However, such diagrams are rather close to the relational approach and not to Chen's model. The number of editors covering the latter is rather limited. One with a good set of concepts is the Enterprise Architect [22], which covers most elements considered prevalent in ER models as of today: n-ary relationships, generalization, and multi-valued or composite attributes. However, there is no transformation approach in place. Only one editor features an ER model like proposed by Chen and also a transformation approach: ERD+ [17]. But that editor only features a rather limited set of modeling elements. It does not support n-ary relationships and the use of generalization is rather restrictive. In addition, the translation of weak types is only possible on the most trivial level. This also applies to the translation of multi-valued and composite attributes. The combination of attribute types is not supported at all. In summary, it can be said that the tool can only be used for simple, not extensive ER Models.

As ER modeling stays relevant, there are also contemporary approaches dealing with this model. However, most of these approaches don't deal with transforming ER models to relational ones. Examples include approaches for creating ER models from text using natural language processing [24]-[26] or applying ER modeling for creating specific models, e.g., for ontologies [27], Kanban systems [28] or software structures [29]. A small number of approaches deals with the relational transformation, but they either provide only very basic transformations with no practical application [30] or no novel transformations at all [31].

All in all, it can be stated that the proposed transformation approaches did not make it into applicable tools, mostly due to the lack of coverage of operational issues. On the other hand, modern editors seem to focus on simplified binary variants that are closer to relational tables than to Chen's ER model.

III. Definition of the ER Model

As described in Sections I and II, there is a number of variations of ER models. However, in order to develop algorithms for transforming the ER model to the relational model, it is mandatory that modeling capabilities are known. Due to this, the ER model used in this work will be defined to ensure an unambiguous transformation.

When defining the modeling capabilities, one goal is to provide the modeler as much freedom as possible. The ER model defined in this paper is heavily based on the model presented by Kemper and Eickler [32]. This model includes the most prevalent modeling components such as entities, n-ary

relations and attributes. It also contains existence-dependent types and covers generalization in form of IsA-Structures. The model is additionally extended by the attribute types ”Multi-valued attribute” and ”Compound attribute” according to Vossen [33].

An entity is the most basic ER component, covered within this paper. It does not have a direct connection with another entity, must have at least one identifying attribute and may be referenced by any number of attributes.

The relation is derived analogously to the entities from the basic principles of the ER model. Within this work, n-ary relations are supported. Every relation must connect at least two entity types and may be referenced by any number of attributes. To increase the modeling capabilities, the ER model also allows reflexive relations from one entity to itself.

The third component commonly used in ER models are attributes. In the context of this work, a distinction is made between three types of attributes. Regular single-valued attributes, multi-valued attributes and identifying attributes. To extend the capabilities of the model, regular and multi-valued attributes can be referenced by further regular and multi-valued attributes. This means that each regular and multi-valued attribute can act as a composite of other attributes. The attributes that form a composite are called composite attributes. Attributes, which are part of a composite attribute are also implicitly given the possibility to act as an composite attribute, which allows the multiple application of composition and multi-valuedness. With the multiple application of multi-valuedness and composition, complex attribute structures can be formed. These structures can form cyclic dependencies and ambiguities between attributes when they reference each other directly or indirectly via further attributes. Nevertheless, it must be guaranteed that each attribute can be uniquely assigned to exactly one entity or relationship. Each attribute can be assigned unambiguously to one entity or relationship if there is a direct connection or exactly one path to an entity or relationship via further attributes. The uniqueness requirement implies that attribute structures must have the form of trees with the corresponding entity or relation as root.

The transformation presented in this paper also supports existence-dependent types, which are also referred to as weak types. Weak types depend on a parent type for identification. In the ER model, this dependency is modeled by the use of a relationship between the parent entity and the existence-dependent entity, called a weak relation. It should be noted that in this ER model, the parent does not have to be a strong entity. Therefore, it is possible that a parent entity is also a weak entity, which in turn depends on another parent entity. Due to this multiple application of existence dependence, restrictions are required to uniquely assign a weak entity its dependent type. These are conceptionally analogous to the multiple application of the composition and multi-valuedness of attributes.

The ER model and transformation also incorporates generalization. Overall, generalization can be divided into a number of different types, characteristics, and constraints. In the context of this paper, generalization in the ER model is implemented exclusively by means of IsA structures, omitting the notation and transfer of specific properties of generalization. IsA structures associate multiple entities with each other. Each entity of the IsA structure acts as a subtype or supertype entity. An IsA structure references exactly one supertype entity and any number of subtype entities. The subtype entities of the IsA structure inherit all attributes of the supertype entity.

When defining the rules for IsA structures, a multiple-inheritance of the attributes of a supertype entity to a subtype entity must be excluded. To avoid this, a restriction to tree-structures, as with attributes, could be made but would be too restrictive. Instead the restriction is made based on the three following sets.

A) The ”subtype set” of an entity contains the entity itself and all entities, which inherit from the entity.
B) The ”supertype set” of an entity includes the entity itself and all other entities, which the entity inherits from.
C) The ”influenced type set” of an entity includes the subtype set and additionally for each entity in the subtype set the supertype set.

The sets defined above on the basis of an expression of several IsA structures are shown in Figure 1. The sets here start from the entity highlighted in blue. The entity within the area shown in red is part of the supertype set. The entities within the green area shown are part of the subtype set. The influenced type set contains all entities, which are highlighted in purple.

If an entity is connected as a subtype of an IsA structure the influenced type set of the entity must be disjoint with the supertype set of the supertype of the IsA structure. In contrast, when an entity is connected as the supertype of an IsA structure, the influenced type set of all subtypes of the IsA structure must be disjoint with the supertype set of the entity.

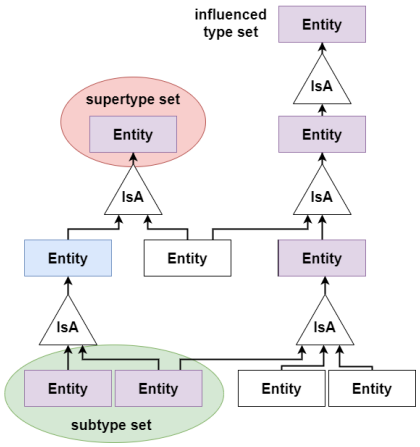

Fig. 1. Subtype-, supertype and influenced type set

In summary, the ER model discussed in this paper is extended by the following concepts:

- Multivalence of attributes

- Composition of attributes
- Reflexive, unary relationships
- N-ary relationships
- Existence dependency
- Generalization

IV. Transformation

The implementable transformation of ER models is realized in several steps, which are executed sequentially.

A) Creating a data model structure of ER diagrams
B) Transformation of attributes
C) Transformation of IsA structures
D) Transformation of weak types
E) Transformation of relationships
F) Cascading of primary keys for attributes

These steps will be explained in more detail below.

A. Structural data model of ER models

In order to execute algorithmic approaches for translating the ER model into the relational model, a basic data model is required on which they can operate.

Any ER diagram essentially consists of elements such as entities, relationships, attributes, IsA structures, and associations between those elements. Therefore, the structure can be expressed directly as a graph. Furthermore, information about the cardinality between a relation and an entity can be stored in the edges of the graph. In case of a IsA structure, the edges also contain information about whether the connected node is a supertype or subtype.

In Section III, it was explained that attributes are always expressed in the form of trees. This makes it possible to further restrict the graph. Since all entities and relationships can have attributes, each of these elements acts as the root of a tree. Each attribute in the ER model is therefore represented as a node in the tree. The edges within the tree structure do not hold any additional information.

B. Transformation of attributes

The goal of this translation is to express any attribute structure by means of relations. From the previous subsection, it is known that all attributes are held within a tree. Therefore, only these trees have to be considered in this translation. If a tree consisting exclusively of single-valued attributes is considered, this can be solved trivially by creating a relation for the tree's root entity or relationship and adding each attribute, which is a leaf of the tree to that relation. Algorithmically, this can be done by traversing the tree in post-order and checking whether the current node is a leaf. However, if a tree contains multi-valued attributes, these cannot be added to the tree's root relation as a relation expresses a fixed sized schema and multi-valued attributes can take on any number of values. In this case, a standalone relation must be created for that attribute and a reference between it and the tree's root relation has to be created. In the case of composite attribute structures, attention must be paid between which relations a reference is created. It is important to note, that the referenced relation does not necessarily have to be the tree's root relation. As it is quite possible that the relation of a multi-valued attribute references a relation of another multi-valued attribute. The transformation of composite attribute structures can be executed by creating a relation for each attribute at the beginning. If the tree is then traversed in post-order, and the current node represents a multi-valued attribute a reference can be created between the current node's relation and the node's parent relation.

Listing 1. Transformation of attributes

```
1  Function TransformAttributeTree (Parent)
2      For Each Child in TreeNode //Execute post-order
3          TransformAttributeTree (Child)
4      End For
5      If TreeNode is Multivalued Attribute Then
6          TreeNode.Table <- marked
7      End If
8
9      If TreeNode is Leaf Then //Recursion resolution
10         Return
11     End If
12
13     For Each Child in TreeNode
14         If Child.Table is marked Then
15             If TreeNode.Children.Size = 1 Then
16                 //Handling of "forwarding" attributes
17                 MergeTable(TreeNode.Table, Child.Table)
18                 TreeNode.Table <- marked
19             Else
20               TreeNode.Table.References.Add(Child.Table)
21             End If
22         Else
23             MergeTable(TreeNode.Table, Child.Table)
24         End If
25     End For
26 End Function
27
28 Function MergeTable(ParentTable, ChildTable)
29     ParentTable.Columns.AddAll(ChildTable.Columns)
30     ParentTable.References.AddAll(ChildTable.References)
31     Delete ChildTable
32 End Function
```

For single-valued attributes, the relation can be merged with the relation of its parent node. Within the merge, all attributes and references of the child relation are transferred to the relation of the parent node. This procedure can be extended by skipping composite attributes, which consist of only one additional multi-valued attribute.

Listing 1 shows the algorithm for translating attributes in pseudo code. It is executed for each entity and each relation in the ER graph. The algorithm is explained below.

The initial situation of the algorithm, shown in Listing 1, is that the ER graph has been created. In addition, a relation is created for each attribute. As within this algorithm, relations which correspond to a single-valued attribute are successively unified. The relations which were created for multi-valued attributes are preserved. For these relations only the above mentioned references are created. In the shown algorithm the postorder traversal takes place in the lines 2 to 4, as well as 9 to 11. For the handling of multi-valued attributes, these are marked in each call in lines 5 to 7. This takes place before the recursion resolution, in order to seize also multi-valued attributes, which are leaves of the attribute tree. Otherwise, those would not be marked and would be merged in the following lines. In lines 13 to 25, each direct child attribute is handled for an attribute. If it is a marked attribute, a reference

to the child attribute is created in line 20. In the special case that the attribute has only one multi-valued attribute as a child, lines 15 to 18 are executed and a ”skipping” takes place, regardless of whether the current attribute is a multi-valued or single-valued attribute. If, on the other hand, it is a single-value attribute (line 23), the relation of the child attribute can be resolved by merging it with the relation of the current attribute. The merging itself is done by adding all columns and references of the child table to the parent table (lines 29 to 31).

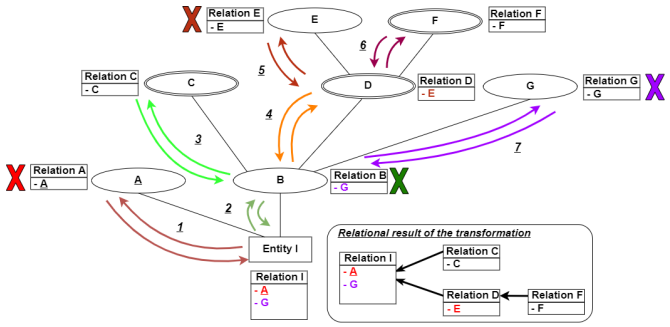

Fig. 2. Transformation of complex attribute structures

Figure 2 illustrates this algorithmic process. The arrows indicate, how the algorithm will process the attributes. The cross next to the relations shows that these have been resolved. The resolution takes place in the same call as the arrows shown in the same color. The result is shown on the bottom right. Note that the attribute value D consists here of many attribute values F and one attribute value E. A direct reference between the entity relation and the relation F could not represent this situation. Also note that the attribute B is merged within the algorithm. This is permissible because entity I is associated with exactly one value for the attribute G.

To complete the translation of the attributes into the relational model, the references between the created relations must be mapped in the form of foreign key dependencies. For this purpose, the remaining relations can be traversed in pre-order and primary keys can be added to the relations. These also act as foreign keys to the primary keys of the referenced relation. The cascading of the foreign keys is not executed directly, as it is only ensured that the root entity or relation contains all primary keys due to its identifying attributes. However, as new primary keys may be added if the entity is a weak entity or part of an IsA structure the immediate execution could lead to invalid references between the relations. Therefore, the execution of the cascading will take place at the end of the whole transformation process as Step F.

C. Transformation of IsA-Structures

The transformation of IsA structures is realized by means of foreign key dependencies between the subtypes and supertypes of the IsA structure. It should be noted that the relations of the entities must exist and the primary keys must be located in them. Because of this, the transformation of the attributes must take place before the transformation of IsA structures. Each subtype of an IsA structure inherits all primary keys of the supertype. In addition, these inherited primary keys refer to the upper type as foreign keys. If entities are part of several IsA structures, ”higher level” IsA structures have to be translated first to ensure that entities at lower levels receive all primary keys. To illustrate the translation order Figure 3 shows an entity-relationship model on the left and the relational model on the right. The red highlighted IsA structure can only be translated after the blue and green IsA structures. The blue one, on the other hand, only after the green one above it.

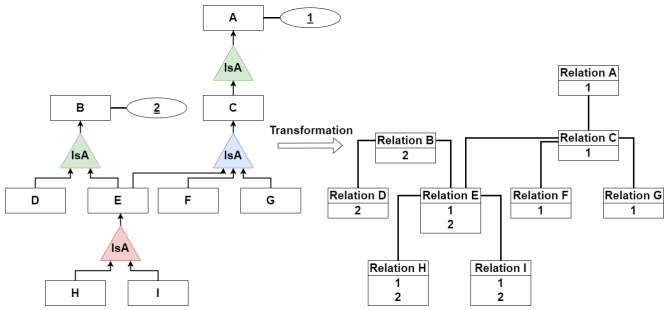

Fig. 3. Transformation order of IsA-Structures

Listing 2 shows the algorithm for transforming an IsA structure. Here, the processing order of the structures is maintained by traversing all IsA structures up to N times, where N is the number of IsA structures.

If the supertype of the selected IsA structure inherits from further IsA structures, and these have not yet been translated, the current pass is skipped (lines 6 to 10). Specifically, line 6 determines all IsA structures that must already be transformed in order to transform the current IsA structure. In Figure 3, this corresponds to the green and blue highlighted IsA structure in the case of the currently treated red IsA structure. Following this, line 7 checks whether all have already been transformed. If at least one IsA structure has not been transformed, the current call is skipped. This also applies if the selected IsA structure has already been translated (lines 2 to 4). If the IsA structure can be transformed in this call, the actual transformation takes place by creating the foreign key dependencies in lines 12 to 16.

Listing 2. Transformation of IsA-Structures

```
1  Function TransformIsAStructure(IsAStruct)
2    If IsAStruct is transformed Then
3      Return
4    End If
5
6    UpperLayerIsAs <- GetInheritedIsAs(IsAStruct.SuperType)
7    UnhandledUpperLayerIsAs <- UpperLayerIsAs !transformed
8    If UnhandledUpperLayerIsAs not empty Then
9      Return
10   End If
11
12   For Each SubType in IsAStruct.Subtypes
13     For Each PrimaryKey in SuperType
14       AddForeignKeyAsPrimaryKey(Supertype, Subtype)
15     End For
16   End For
17   IsAStruct <- isTransformed
18 End Function
```

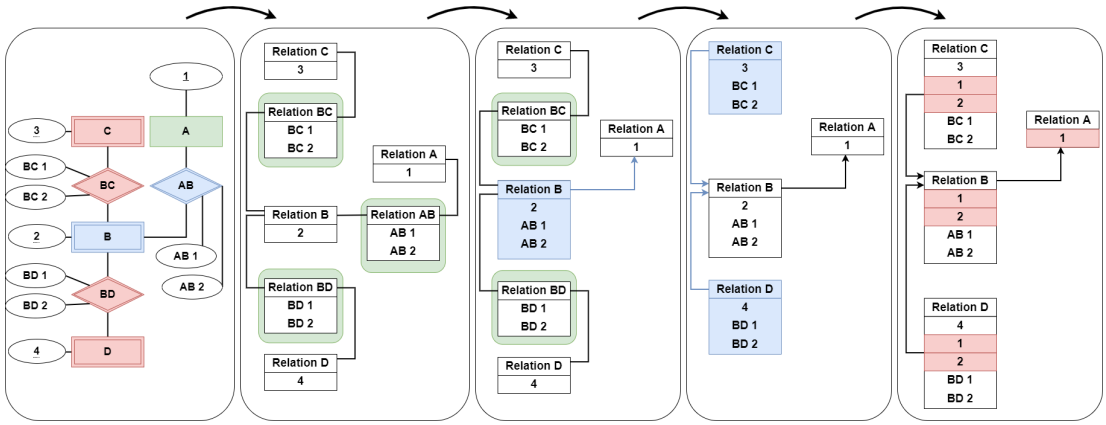

Fig. 4. Transformation of weak types

D. Transformation of weak types

For the translation of weak types to the relational model, it is a prerequisite that relations exist for all entities and relationships and that all attributes are already contained in them. Therefore, the transformation of attributes and IsA structures must be performed beforehand. IsA structures must be translated before, since a strong entity, on which a weak entity depends, can receive further primary keys during the translation of IsA-Structures.

Equivalent to the transformation of IsA-Structures, the translation order has to be considered. The translation has to start from weak entities, which have a connection to a strong entity or an already translated weak entity by means of a weak relationship. The relation of the weak relationship always has to be merged with the relation of the dependent entity. During the translation, the relation of the entity to be translated keeps a reference to the strong or already translated entity. This reference can then be used to create the foreign key dependencies.

The translation process of an ER diagram is shown in Figure 4. The figure starts immediately after the execution of the algorithms for the translation of attributes and IsA structures. The first rectangle shows an example ER model, which is transformed over a series of steps. The second step is the starting point of the algorithm, where all elements occur as a relation, conditioned by the previously executed attribute algorithm. The green highlighted elements represent the weak relationships in the ER model, which are required to be transformed. According to the mentioned translation order, the elements to be translated are determined and transformed in each step. In Figure 4, the blue elements are translated first, followed by the red elements.

Algorithmically, the merging of relations and reference creation from Figure 4 is shown in Listing 3. The compliance with the order is done, analogous to the translation of IsA structures, by checking all weak entities up to N times, where N equals the number of weak entities in the graph.

Listing 3. Transformation of weak types

```
1   Function TransformWeakEntity(WeakEnt)
2       If WeakEnt is transformed Then
3           Return
4       End If
5
6       WeakRelations <- GetConnectedWeakRelations(WeakEnt)
7       For Each WeakRel in WeakRelations
8
9           OtherEnt <- GetOtherEntity(WeakEnt, WeakRel)
10          If OtherEnt is no StrongEntity or
11                      is not transformed WeakEntity Then
12              Continue
13          End If
14          WeakEnt.Transformed <- true
15          WeakEnt.References.Add(OtherEntity)
16          MergeTables(WeakEnt, WeakRel)
17          Return
18      End For
19  End Function
```

In contrast to the IsA structure algorithm, the transformation algorithm resolves all connected weak relationships (line 6) and immediately tries to transform them (lines 7 to 18). If weak relationships are connected to a strong entity or an already transformed weak entity (lines 9 to 13), the current weak entity can determine its existence-dependent type and can be transformed (lines 15 to 16).

Note that the cardinalities of the weak relationship are not to be considered for the basic transformation, since these can only be 1:1 and N:1 towards the identifying type. The given algorithm realizes both functionalities by means of a foreign key dependency.

E. Transformation of relationships

Transformation of regular relationships requires prior execution of all previous algorithms, since all relations for entities, weak types and relationships require to have the complete primary keys.

Generally, there are three cases to consider when translating.

If a relationship connects two entities and the cardinality is N:M, the primary keys of the two entities are added to the relation of the relationship. These then reference the primary keys of the entity relations as foreign keys. The translation of N-ary relations is done regardless of their cardinality. The transformation of these is analogous to the translation of binary N:M relations. In this case, the relation of the relationship receives the primary keys of all connected entities. Each of this

primary keys refers to the primary key of the corresponding entity in the form of a foreign key.

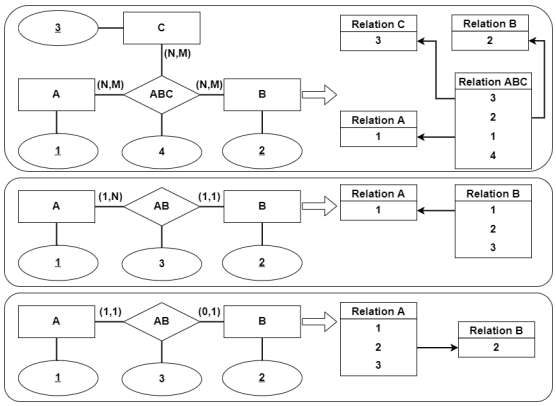

Fig. 5. Transformation of relationships

If the cardinality of the relationship is 1:N, the relationship is resolved by merging the relation of the relationship with the entities relation on the N side. In addition, this merged relation receives all primary keys of the opposite entity as normal attributes. These act as foreign keys on the opposite entity. In the third case, the relationship cardinality is 1:1. The translation is to be performed analogously to 1:N relationships, and the entity that receives the foreign keys and relation attributes must be specified for this purpose. Since this work, the Min-Max notation is used, the optionality has to be considered to avoid zero values. If one of the cardinalities describes an optionality, the attributes of the relationship and foreign keys are added to the entity on the other side. If both or none of the cardinalities describe an optionality, then the optionality is arbitrary.

Figure 5 shows the above cases. On the left side is the ER model. On the right side is the relational model resulting from the translation of the ER model. The first box in Figure 5 shows the transformation of N:M relationships, while the second displays a 1:N transformation. The last one shows the transformation of an optional 1:1 relationship.

V. Practical implementation

Since the main focus of this paper is on practical application, we also provide an implementation of our concepts. To evaluate the algorithms given in this work a web application has been developed and put into operation, which implements all algorithms presented in this work. The application also contains a graphical editor.

Within the editor the user is able to create extensive and complex ER models, which are only restricted by a few limiting rules, required to allow an unambiguous logical assignment of the ER components. Based on the user-generated diagram, the presented algorithms are used to transform the diagram into a relational model without further user intervention.

To enforce correctly modeled ER diagrams, a validation process was implemented. This validation mechanism is designed in a user-friendly and proactive manner to ensure that a user is able to model fast and easy. Therefore, the validation offers maximum flexibility by only restricting actions which would necessarily lead to a violation of a rule and thus to an mandatory reverse action. Utilizing the presented algorithms and the validation procedure, the graphical editor is able to translate any model that can be modeled with it into the relational model. Furthermore, the editor visually presents the created relational model to the user.

To increase the practical applicability, an SQL generator was implemented within the application, which works on the basis of the generated relational model and generates SQL schema definitions. The generated SQL is in PostgreSQL dialect.

The graphical editor is shown in Figure 6. At first, a left side bar can be seen. Elements on this bar can be dragged into the drawing area to the right to create new elements. The use of drag & drop is intended to make it possible to create elements quickly and intuitively. The drawing area itself can be expanded endlessly to the right and bottom. If an element is selected, a side bar becomes visible on the right side, which provides additional options. These are, for example, assigning a name, deleting the element or creating a link to another element. In the case of Figure 6, a relation was selected which further enables the option to add new associations or edit the cardinalities to existing entities.

To further enhance the practical applicability, a save and load function has been implemented. Using this functionality by clicking buttons on the right top of the editor, a model can be saved in the form of a text file and at any time be loaded into the editor. By using the button at the center bottom the model can be transformed into the relational model, which will be, without the need of any further user interaction, transformed by the implemented algorithms and visually presented to the user. Furthermore, it is possible to freely switch between the conceptual and relational view using the tab bar in the upper left. In order to generate SQL code from the relational model, data types can be entered in the columns of the relational model.

The translation process from the ER model to the relational model has already been explained in detail. However, to make the transformation practically applicable and the editor effectively usable, two further building blocks have been integrated as mentioned before: The validation of the ER models created by the users and the automatic generation of SQL based on the generated relational model. Both of these will be described in the following.

A. Validation

An essential aspect for the practical applicability of the ER modeling tool, or ER modeling using a tool in general, is to ensure in a very user-friendly way that the ER model created within it complies with the definition for ER models given in Section III.

Several mechanisms to ensure compliance with the ER modeling rules have been explored. These range from native approaches such as holistic validation of the model at the time of translation, to approaches that take effect within the

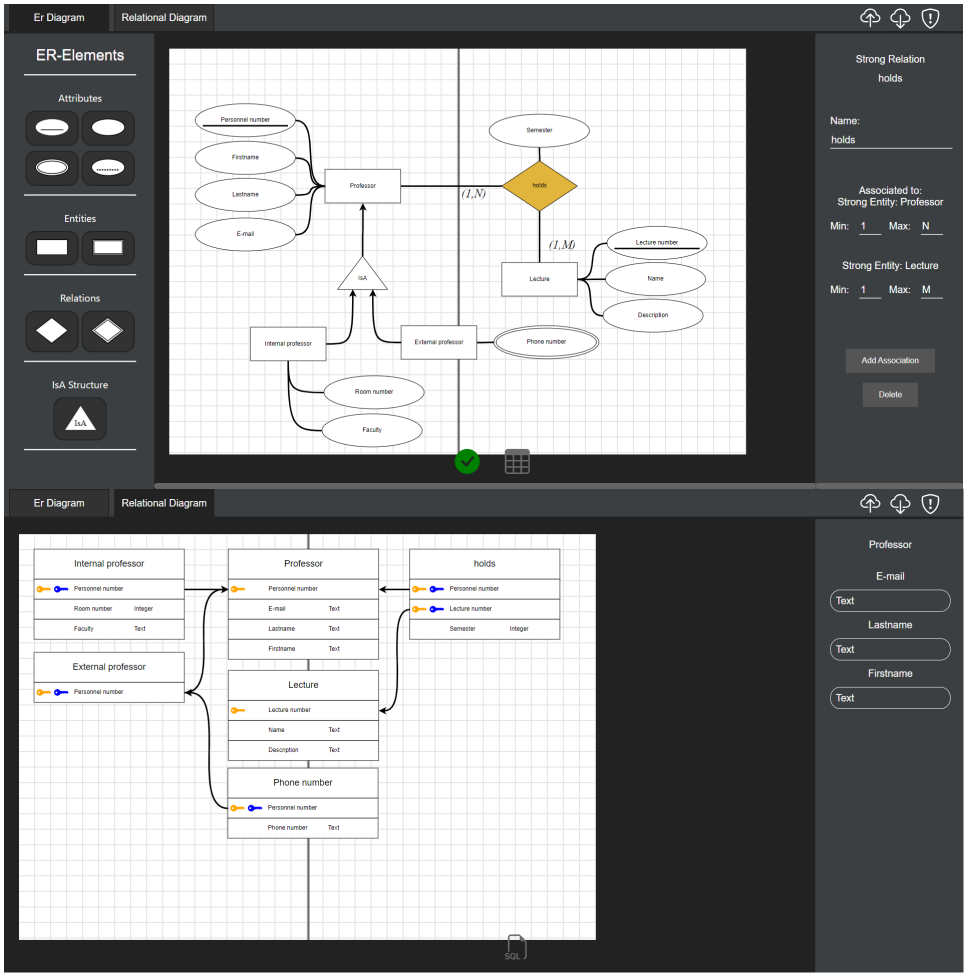

Fig. 6. ER Modeling Tool

modeling process and restrict user options in advance that could lead to an invalid ER Model.

The approach most suitable for this tool is briefly presented below. The approach can be described as an extended variant of the Correctness by Construction principle [34].

The basic idea of this validation system is to proactively prevent actions within the modeling process that would inevitably lead to a violation of a rule. In contrast to a pure Correctness by Construction approach, the idea is extended by not only allowing actions that transition the ER model from one valid state to another, like adding a new entity to the model. The extension involves the introduction of an additional state to allow the user much more flexibility during the modeling process.

To understand exactly what restrictions would be involved if the additional state is not used, an example modeling process is shown in Figure 7.

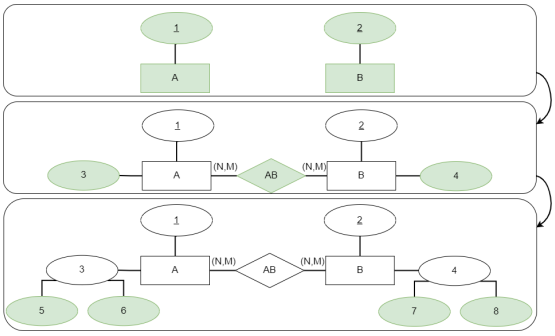

Fig. 7. Modeling process with simple valid to valid state transitions

The boxes shown in Figure 7 show the gradual creation of the ER model across the valid states. The elements shown in green are the newly added ones. It shows up clearly that if every action of the user would be checked for compliance with the rules, the user would be strongly limited in his possibilities. For example, it would not possible to freely delete elements or to model attributes first.

Modeling using the new state allows these restrictions to be avoided.

The additional state in which the ER model can be is defined as follows: An entity-relationship model is in a partially consistent state if it can be transformed by insertion operations, consisting of adding elements and associations, into a state that satisfies all ER rules defined in Section III.

This definition of a state of an ER diagram is a subset of the possible valid and invalid forms of an ER diagram. The state implicitly assures through its definition that it can be transformed into a valid ER model, but still gives the modeler a lot of freedom.

For example, modeling using this state does not restrict the insertion of elements, since ER models are generally not finite and each element can be integrated into an ER model at any time. In addition, deleting elements is not restricted in any way. If the original state of the ER diagram is in a partially consistent state, all ER diagrams that are changed by deletion operations are also in a partially consistent state, since they can be restored to their original state by a reverse insert operation.

This state practically only restricts operations that would force the ER model into an invalid state, such as an association of two entities without a relationship. In order to restore this model to a valid state, it would be necessary to remove exactly this association. Using this state, the modeler is given the highest possible freedom in modeling without compromising the consistency of the ER model.

In practice, the check is performed with every action of the modeler. For example, if an element is selected, all other elements of the model are checked one after the other. During the check, an association is created between one of the elements and the selected element and it is examined whether the model still corresponds to the new state. If this is the case, an association between these two elements is permitted. In Figure 6, this would be indicated by the permissible elements being highlighted.

Visually, the gradual creation of the er model with this state is exemplified in Figure 8. The boxes and elements highlighted in green can be interpreted in the same way as in Figure 7, where the elements shown in green are the newly created ones.

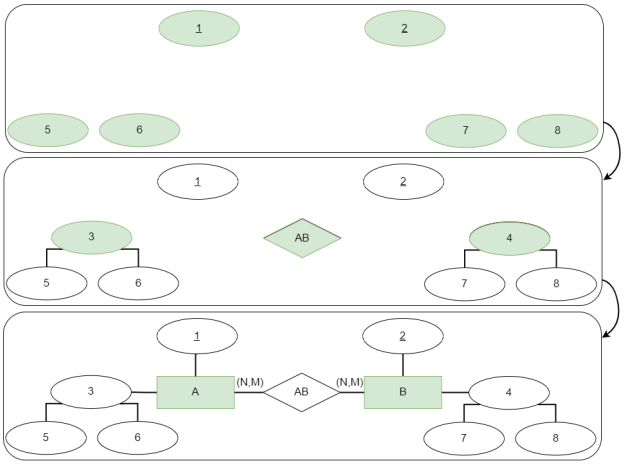

Fig. 8. Modeling process with transitions to additional state

However, Figure 8 also shows that the ER diagram can be far from a valid state. Therefore, a feedback system was installed, which informs the user at which parts the model is not yet fully valid by checking the current state of the er model against a valid one.

B. SQL Generation

This section provides a brief summary of the feature for generating SQL from the relational model. In principle, the relational model generated from the ER model can be represented as a graph in which tables form the nodes and foreign key constraints form the edges between the nodes. An example is shown in Figure 9. On the left side a relational model is shown. The right side, in turn, shows the resulting directed graph.

A topological sort is performed on these graphs. Thus, the result of the algorithm is an ordered sequence of tables, where each table occurs before it is referenced through foreign key constraints of other tables. SQL code can then be generated for each table using an SQL template. These statements are combined afterwards to achieve an SQL script for the whole model. This allows a sequence of SQL statements to be generated for a relational model, which can be executed immediately one after the other, thus enabling the modeler to implement the relational model in a target database.

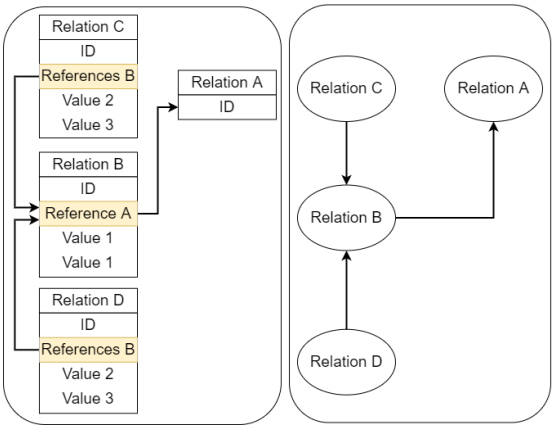

Fig. 9. Relational model as graph

VI. Evaluation

In this paper we proposed a transformation of ER models to relational models with focus on practical applicability. Therefore, we also chose a practical kind of evaluation, which is two-fold.

On the one hand, we implemented the editor tool described in the previous section. In its current state, the editor allows for ER modeling and an automated transformation of all mentioned concepts. By means of these results, it is shown that the given algorithms are implementable and are capable of performing a transformation of ER models into relational models. Using this editor, we also conducted a preliminary evaluation regarding the correctness of the transformation by testing the editor with a predefined set of ER models

containing different combinations of modeling elements. The evaluation was successful as the editor correctly transformed all supported concepts. To enable a broader evaluation and application of the editor, we made it available open source [35].

On the other hand, to gather empirical evidence about the capability of the transformation and the functionality of the editor, we conducted a practical evaluation with 15 undergraduate computer science students of the third semester. All of the subjects attended the lecture "Database Systems", five of them had prior database knowledge. For the evaluation, they had to create two ER models from a textual description. Both of these were also part of the lecture and the students had already completed the tasks two months before as exercises with an editor of their choice. That way, we could also obtain comparative information regarding our editor and others. The modeling tasks included all concepts relevant for the database lecture: Entities, different types of relationships, different types of attributes, and inheritance. After completion of the tasks, the subjects had to fill out a questionnaire. The results will be discussed in the following while Appendix A contains several figures summarizing the results.

The first question dealt with the intuitiveness of the editor. The subjects had to answer with a score from 1 (very bad) to 5 (very good). The average of the answers (cf. Figure 10) of the 15 subjects was a score of 4.13. The subjects also stated that none of the necessary modeling elements were missing (cf. Figure 11). Optional comments regarding missing features of the editor targeted usability improvements (multi selection of elements and shortcuts for commands). However, the provided modeling elements did mostly behave as the subjects expected (cf. Figure 12). There were also optional comments about behavior of the editor/elements that should be improved:

- Editing pane too small on small displays 1
- No non-integer primary keys 1
- Resizing of entities, relationships, and attributes 1
- N-ary relationships 1
- Creating relationships is cumbersome 1

The subjects were also asked about their previous editor of choice and provided the following answers:

- Enterprise Architect 7
- Draw.io 3
- ERDplus 1
- None 1
- Metatron 1
- Miro 1
- Visio 1

Comparing the modeling experience of the other editor to the one presented in this paper, the majority of the subjects preferred the latter (cf. Figure 13). In addition, most of the subjects would prefer our editor or a combination in the future (cf. Figure 14). When asked for the advantages of our editor, the subjects provided the following answers:

- Conversion to database schema 5
- Simplicity 5
- Intuitiveness 4
- Modeling support 2
- Contains all necessary elements 2
- Quicker modeling 1

The subjects also provided information about disadvantages of our editor compared to the prevalent ones:

- GUI visual representation 6
- Editing of nodes cumbersome 1
- No automatic saving 1
- Draw area too small 1

Besides the editor, the evaluation also focused on the implemented transformations to the relational model and to SQL. Nearly all of the subjects stated that the transformation to the relational model worked (cf. Figure 15). However, it turned out that the problems two subjects had with the transformation were caused by a visual problem: The current state of the editor only supports one browser. If executed in another, the relational visualization has some issues. This was also stated by the users when asked for problems regarding the transformation:

- Visualization problem 4
- Data types missing in relational model 1

Due to all subjects, the transformation to SQL worked well (cf. Figure 16). The same applied for the correctness of the created models (cf. Figure 17). When comparing the transformation of the editor to their previously created own manual transformation nearly all subjects rated the one of the editor equally well or better (cf. Figure 18).

In summary it can be stated that the implemented transformations worked nearly perfect for all of the subjects in this practical evaluation. The editor software was also perceived rather positive. However, for the latter there was also constructive criticism, mainly relating to usability. We will take this into account and use it for improving the editor in future versions.

VII. Conclusion

Despite its age, ER modeling is still the most prevalent way of creating conceptual data base schemata. Since its advent in the 1970s, various extensions have been proposed. Due to this, many different flavors are currently used in modern editors. To be applicable as technical database schema, the ER models have to be transformed into relational models. This can be a complicated and error-prone task. Therefore, various standardized transformation approaches have been proposed over the decades. However, these approaches remained rather theoretic and did not include operational semantics. Thus, no tool support was established utilizing them and the transformation process remained manually to a large extend.

Despite this issue, modeling support for ER models was achieved. To date, a high number of editors is available in different flavors. There are diagram tools offering ER diagram creation, database clients with ER schema creation options, or other editors like UML editors incorporating database modeling. While some of them only provide diagrams, others

enable the direct application to relational databases. However, there is still no practical automated transformation of ER diagrams to relational ones. Editors offering this do not enable the modeling of real ER diagrams but rather enhanced DB diagrams or omit important prevalent concepts like n-ary relationships or generalization.

To tackle this issue, we proposed an approach for transforming ER models to relational ones with a strong focus on applicability and operational issues. The ER model incorporates the most prevalent and necessary concepts [32] as n-ary relationships, multi-valued and composite attributes, or generalization. All of these can be correctly transformed in any meaningful combination enabling great flexibility for the input models.

To prove the applicability of the proposed approach, we have implemented a graphical ER editor capable of creating diagrams containing all mentioned concepts as well as a an automated transformation to relational models. As practical applicability was our focus, we also added a validation mechanism to the editor that guarantees the creation of correct and transformable ER models while providing the user as much flexibility as possible and a good user experience. Furthermore, we have conducted an evaluation study with 15 subjects investigating the correctness and applicability of our approach as well as the usability of the proposed editor. The findings of this study include that the approach works well and is usable within the created editor. Comments for improvement targeted only usability issues of the editor. The editor is publicly available for further testing and improvement [35]. All in all we have shown a transformation approach that can easily be implemented in editors. This can aid future conceptual database modeling, spare time, and reduce errors resulting from manual transformation.

Our future work will focus on promoting the transformation approach and the editor. We will use the findings of our study to improve the usability of the editor and make it appealing to larger user numbers. This enables future evaluations with broader audiences. Further, we will add additional features to the approach as well as the editor. This includes additional and optional concepts to the ER models, like composition structures as well as other diagram types and transformations.

References

[1] G. Grambow and S. Ruttman, "A Practical Automated Transformation of Entity Relationship Models to Relational Models," 15th Int'l Conf on Advances in Databases, Knowledge, and Data Applications, pp. 5–12, 2023.

[2] P. Chen, "The entity-relationship model—toward a unified view of data," ACM Trans. on DB Sys. vol. 1, no. 1, pp. 9–36, 1976.

[3] R. Elmasri and S.B. Navathe, Fundamentals of Database Systems. Benjamin/Cummings, Redwood City, 1994.

[4] S. Spaccapietra and C. Parent, "ERC + : an object based entity-relationship approach," Conceptual Modeling DBs and Case: An Integrated View of Information Systems Development, John Wiley, 1992.

[5] B. Thalheim, "Extending the entity-relationship model for a high level, theory-based database design," 1st Int. East West Database Workshop, pp. 161–184, 1990.

[6] T.J. Teorey, "Database Modeling and Design," The Entity-Relationship Approach, Morgan-Kaufmann, San Francisco, 1990.

[7] J. Makowsky, V. Markowitz, and N. Rotics, "Entity-relationship consistency for relational schemas," Int. Conf. on Database Theory, pp. 306–322, 1986.

[8] A. D'Atri and D, Saccà, "Equivalence and Mapping of Database Schemes," VLDB '84, pp. 187–195, 1984.

[9] E. Wong and R. Katz, "Logical design and schema conversion for relational and DBTG databases," Int. Conf. on ER Approach to Sys. Analysis and Design, pp. 311–321, 1979.

[10] S. Jajodia, P.A. Ng, and F.N. Springsteel, "The Problem of Equivalence for Entity-Relationship Diagrams," IEEE Trans. on SE vol. 9, no. 5, pp. 617–630, 1983.

[11] T. Teorey, D. Yang, and J. Fry, "A logical design methodology for relational databases using the extended entity-relationship model," ACM Computing Surveys (CSUR) vol. 18, no. 2, pp. 197–222, 1986.

[12] T. Ling, "A Normal Form For Entity-Relationship Diagrams," 4th Int. Conf. on the ER approach, pp. 24–35, 1985.

[13] H. Sakai, "Entity-relationship approach to the conceptual schema design," SIGMOD '80, pp. 1–8, 1980.

[14] Lucidchart. Last visited: 2024.05.14. [Online]. Available: https://www.lucidchart.com

[15] Draw.io. Last visited: 2024.05.14. [Online]. Available: https://app.diagrams.net/

[16] VisualParadigm. Last visited: 2024.05.14. [Online]. Available: https://online.visual-paradigm.com

[17] ERDPlus. Last visited: 2024.05.14. [Online]. Available: https://erdplus.com/

[18] pgAdmin. Last visited: 2024.05.14. [Online]. Available: https://www.pgadmin.org/

[19] MySQL Workbench. Last visited: 2024.05.14. [Online]. Available: https://www.mysql.com/products/workbench/

[20] Outsystems. Last visited: 2024.05.18. [Online]. Available: https://www.outsystems.com/

[21] Mendix. Last visited: 2024.05.18. [Online]. Available: https://www.mendix.com/

[22] Enterprise Architect. Last visited: 2024.05.14. [Online]. Available: https://www.sparxsystems.de/

[23] C. Fahrner and G. Vossen, "A survey of database design transformations based on the entity-relationship model," Data and Knowledge Engineering vol. 15, no. 3, pp. 213–250, 1995.

[24] M. Kasra Habib, "On the Automated Entity-Relationship and Schema Design by Natural Language Processing," Int. J. of Engineering and Science vol. 8, no. 11, pp. 42–48, 2019.

[25] P. G. T. H. Kashmira and S. Sumathipala, "Generating Entity Relationship Diagram from Requirement Specification based on NLP," 3rd Int. Conf. on Information Technology Research, pp. 1–4, 2018.

[26] S. Ghosh, P. Mukherjee, B. Chakraborty, and R. Bashar, "Automated Generation of E-R Diagram from a Given Text in Natural Language," Int. Conf. on Machine Learning and Data Engineering, pp. 91–96, 2018.

[27] M. Ahsan Raza, M. Rahmah, S. Raza, A. Noraziah, and R. Abd. Hamid, "A Methodology for Engineering Domain Ontology using Entity Relationship Model," Int. J. of Advanced Computer Science and Applications vol. 10, no. 8, pp. 326–332, 2019.

[28] K. Ľachová and P. Trebuňa, "Modelling of Electronic Kanban System by Using of Entity Relationship Diagrams," Int. Scientific Journal about Logistics vol. 6, no. 3, pp. 63–66, 2019.

[29] A. Ramírez-Noriega, Y. Martínez-Ramírez, J. Chávez Lizárraga, K. Vázquez Niebla, and J. Soto, "A software tool to generate a Model-View-Controller architecture based on the Entity-Relationship Model," 8th Int. Conf. in Software Engineering Research and Innovation, pp. 57–63, 2020.

[30] Y. Liu, X. Zeng, K. Zhang, and Y. Zou, "Transforming Entity-Relationship Diagrams to Relational Schemas Using a Graph Grammar Formalism," IEEE Int. Conf. on Progress in Informatics and Computing, pp. 327–331, 2018.

[31] L. Yang and L. Cao, "The Effect of MySQL Workbench in Teaching Entity-Relationship Diagram (ERD) to Relational Schema Mapping" Int. J. Modern Education and Computer Science vol. 7, pp. 1–12, 2016.

[32] A. Kemper and A. Eickler, Datenbanksysteme. Eine Einführung [English: DBMS. An Introduction] Oldenbourg, 2015.

[33] G. Vossen, Datenmodelle, Datenbanksprachen und Datenbankmanagementsysteme [English: Data Models, Database Languages and DBMS] 5. Aufl. Oldenbourg, 2005.

[34] T. Bordis, T. Runge, A. Kittelmann, and I. Schaefer, "Correctness-by-Construction: An Overview of the CorC Ecosystem," ACM SIGAda Ada Letters vol. 42, no. 2, pp. 75–78, 2023.
[35] Editor Implementation. Last visited: 2024.05.14. [Online]. Available: https://github.com/SimonRuttmann/ERModellingTool

Appendix A. Study Results

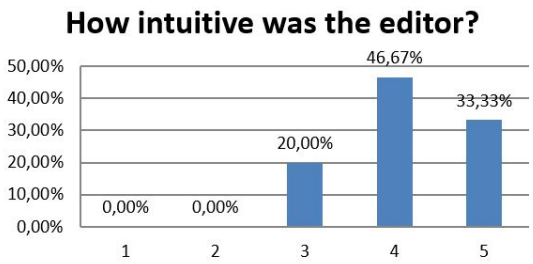

Fig. 10. Intuitiveness of the editor

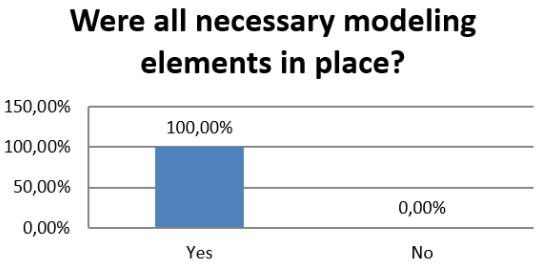

Fig. 11. Modeling elements in place

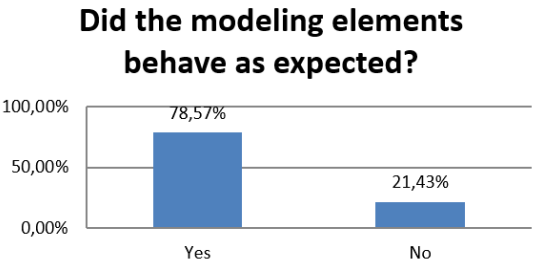

Fig. 12. Elements behave as expected

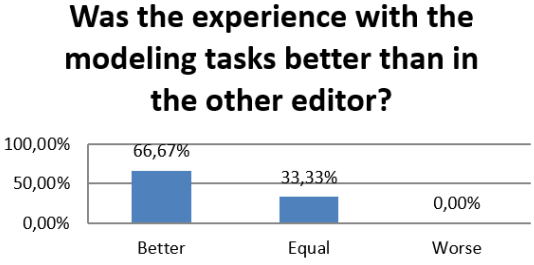

Fig. 13. Modeling experience

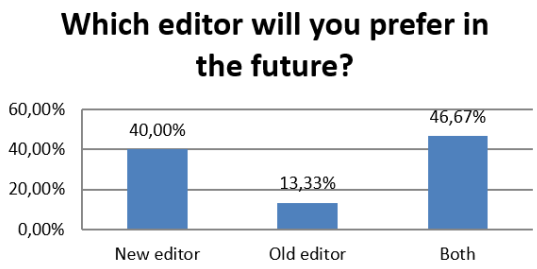

Fig. 14. Editor preference

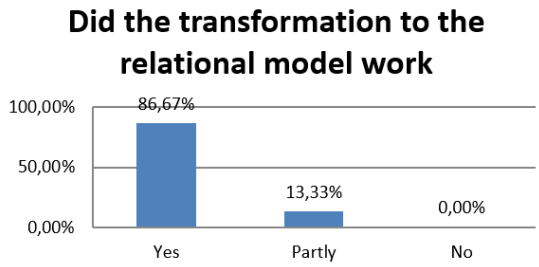

Fig. 15. Transformation to relational model

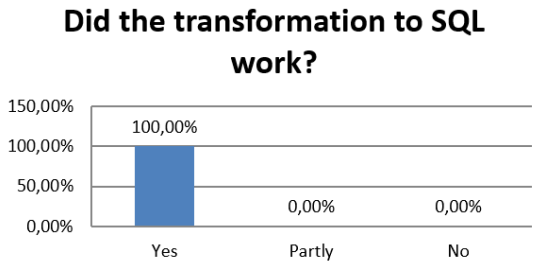

Fig. 16. Transformation to SQL

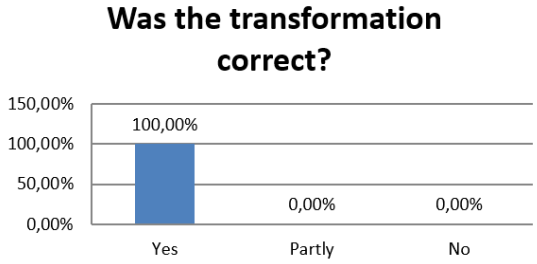

Fig. 17. Transformation correctness

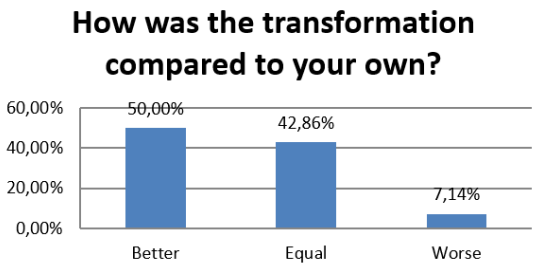

Fig. 18. Transformation comparison