

# Resolving the “Weak Status” of Weak Entity Types in Entity Relationship Schemas

Mira Balaban and Peretz Shoval

Information Systems Program

Dept. of Industrial Engineering and Management

Ben-Gurion University of the Negev

P.O.B. 653, Beer-Sheva 84105

ISRAEL

mira@cs.bgu.ac.il    shoval@bgumail.bgu.ac.il

(972)-7-6472222    (972)-7-6472221

## Abstract

Entity Relationship schemas include *weak* entity types, whose entities are identified by their inter-relationships to other entities. During the translation of the EER schema into a logical database schema, the weak entity types are either, translated into logical units of the database (a relation or a class), or are embedded as attributes of other logical units. Formal studies of EER schemas either ignore the presence of weak entity types, or simplify their dependency structure. The presence of weak entity types in an EER schema may be problematic: A weak entity may not be easy to identify because it may be related to other weak entities in various ways, thus causing problems in schema comprehension, as well as in mapping it to a logical database schema. We claim that the presence of weak entity types in an EER schema belongs to an intermediate design stage, but the final EER schema must not include such entity types.

An algorithm for resolving the status of weak entity types, following user directions, is introduced. If the directions are legal, the algorithm yields an EER schema without weak entity types. The contribution of this paper is in resolving the weak status of weak entity types in EER schemas. The advantage is twofold: First, the translation of an EER schema into a logical database schema can be fully automated. This is essential for upgrading the EER model to support full database management. Second, it enables a fully formal study of the EER model.

# 1 Introduction

ER is a popular visual data model for describing entities, their properties, and inter-relationships. A set of entities that share a common structure is captured as an *Entity Type*. Regular properties of entities are captured as their *Attributes*. Interactions among entities are modeled by *Relationship Types*. Cardinality constraints are set on relationship types, and characterize numerical dependencies among entities within the relationship types. The model was introduced by Chen ([5]), and received many extensions and variations, which are generally termed the Enhanced ER (EER) model. Traditionally, EER schemas are used for conceptual database design, and are translated into logical database schemas, usually relational or object-oriented (OO) schemas ([6]). Application program transactions and consistency checks are added directly to the target database schema.

The EER model is value oriented, in the sense that it assumes that entities in a given entity type are uniquely identified by an attribute (or a set of attributes) of that entity type, called a *key*. Under this assumption, an entity type must have a key, since this is the only means for referencing its entities. However, this assumption is too restrictive since in some entity types entities are identified by their inter-relationships to other entities. For example, in a medical clinic (see Figure 1), a visit is identified, by its date, and by the patient entity that participates in the visit. Such entity types (as **Visit**) are distinguished in EER schemas as *weak* entity types. Their entities are related through *identifying relationship type(s)* to their *owner* entity(s). They are identified by their owner entities, and by their partial key attributes, if any. During the translation of the EER schema into a logical database schema, the weak entity types are either translated into logical units of the database (a relation or a class) with references to the translations of their owner entity types, or are embedded as attributes of other logical units.

The semantics and manipulation of weak entity types becomes complicated, when the owners of a weak entity type are, themselves, weak. In that case, the embedding of a weak entity type as an attribute of the translation of another entity type might be indirect, since the owner entity type might itself be embedded as an attribute. Resolving the embedding in an algorithmic way requires an inductive construction of the *embedding entity type* for a weak entity type. Cyclic embedding constructions are meaningless, and should be declared illegal. When a weak entity type  $E_1$  is translated into a logical unit of the database, while its weak owner  $E_2$  is resolved to be embedded as an attribute of its own (possibly indirect) owner  $E_3$ , the translation of  $E_1$  must, somehow know to reference  $E_3$ .

In general, EER models do not restrict the “ownership structures” of weak entity types. As a result, an EER schema can include complicated *ownership paths*, from a weak entity type to its owners. For example, there can be multiple identification paths between a weak entity type to one of its owners (possibly indirect), or there can be cycles of ownership relationships, etc. The understanding of such structures becomes hard, the unclear semantics turn the schemas incoherent, formal tools that test the consistency and correctness of schemas do not apply to the general case, and mapping algorithms from an EER schema to other schemas do not apply to the general case as well. Consequently, in practice, the embedding of weak entity types is, usually, done manually by the modeler carrying the EER schema translation. Standard translation algorithms fall short of full algorithmic translation. Moreover, formal studies of EER schemas either ignore the presence of weak entity types ([8, 10, 4]), or simplify their dependency structure ([7]).

EER schemas with weak entity types are acceptable as long as the conceptual model is used only in the initial stages of design, for representing and comprehending reality. But in the final stage of design, there should not be weak entity types because of the above mentioned problems. Moreover, conceptual data models have become significant in heterogeneous environments, where they are used as unifying meta-schemas, that coordinate the cooperation of system modules that assume different data models. This status requires that they are formally defined. In particular, the vague status of weak entity types prevents the upgrading of the EER data model into a conceptual DBMS<sup>1</sup>.

We claim that although the presence of weak entity types was practically shown essential for data analysis and design, their status is not well defined. Moreover, they are treated differently in the various ER models. The goal of this work is to clarify the problematics of weak entity types, and suggest a solution that does not exclude weak entity types altogether. We conceive conceptual modeling as a process of construction of conceptual schemas. The process starts with a loosely defined schema, and proceeds towards a well defined one. The elimination of weak entity types, which is the focus of this paper, is part of this process. The final schema includes no weak entity types, and is, semantically, well defined. For that purpose, we introduce an algorithm for resolving the status of weak entity types in EER schemas, within the context of the schema. The algorithm is, in essence, an upgrading of the standard handling of weak entity types to the EER schema level.

---

<sup>1</sup>Indeed, the motivation for this work came from the need to implement a MEER schema (EER with structure Methods [2, 3]) in terms of an object-oriented database schema. The complexity of ownership structures for weak entity types made the translation impractical.

The user marks the weak entity types that are candidates for being embedded as attributes of other entity types. If the embeddings are possible (legal), they are built by the algorithm. Otherwise, the algorithm fails, pointing out the cause of failure. Overall, if the algorithm succeeds, it yields a new EER schema without weak entity types, and with *reference attributes* instead.

The contribution of this paper is in resolving, once and for all, the weak status of weak entity types in EER schemas. The algorithm we introduce can be applied as the last step in the design of the schema. Once this is done, all existing formal investigations of EER schemas are applicable, and EER schemas can be formally defined as an abstract conceptual level on top of a logical database schema. That is, the resulting EER schema can be automatically implemented in terms of a logical database schema. We believe that our approach clearly separates the conceptual modeling stage, where the user must be consulted, from the implementation stage, that should be carried out automatically.

In Section 2 we discuss the issue of weak entity types in EER schemas, emphasizing complex interrelationships between a weak entity type to its owners. In Section 3 the resolution algorithm for weak entity types is introduced and demonstrated. Section 4 is the conclusion.

## 2 Weak Entity Types in the EER Data Model

An EER schema consists of *Entity Type* symbols, *Relationship Type* symbols (each with associated arity), *Role Name* symbols, and *Attribute* symbols. Entity type symbols can be *strong* (denoting non-weak entity types) or *weak*. Attribute symbols can be *simple* or *composite*. A composite attribute symbol is associated with a set of attribute symbols (other than itself). Furthermore, each attribute symbol is either *single-valued* or *multi-valued*.

A key of a type is a means for identifying the instances of the type via their attributes. A key of a strong entity type symbol and of a relationship type (if any) is an attribute of the type. A key of a weak entity type symbol is defined through related *owner* entity type symbols. Every weak entity type symbol  $E$  is associated with binary relationship type symbols that are singled out as the *identifying relationship types* of  $E$ . The entity type symbols associated with  $E$  through these relationship type symbols are termed the *owner entity types* of  $E$ . The cardinality constraints associated with the identifying relationship types of  $E$  guarantee that every entity of  $E$  is related through an identifying relationship type to a single owner entity. The key of  $E$  consists of the keys of its owners, and its own *partial key* (if exists), which is any of its attributes.

For the sake of keeping this paper short, we avoid presenting a formal definition of the EER data model, its semantics and consistency. Such definitions have been provided elsewhere ([9, 1]). Figure 1 presents an EER diagram for a medical clinic. Rectangles describe entity types, diamonds describe relationship types, circles describe attributes, Solid lines among rectangles describe entity type hierarchies, with arrow heads pointing to super entity types, and dotted line rectangles and diamonds stand for weak entity types and their identifying relationship types (with arrow heads pointing to owner entity types).

In Figure 1 we see that **Lab-test-prescribed** is a weak entity type owned by the **Lab-test** and the **Visit** entity types. **Visit** is itself weak; it is owned by **Patient**, and identified by the **Patient Id**, and the **Visit date**. The **Visit** entity type stands in a regular relationship with **Physician**. If **Visit** is resolved to be embedded as an attribute of its owner **Patient**, then whoever references **Visit**, needs to know its **target\_entity\_type**, i.e., **Patient**. This applies to the **treating** and the **prescription** relationship types. Hence, every embedding requires the specification of the final **target\_entity\_type**. The schema might include also a weak entity type with multiple owner entity types. For example, **Lab-test-prescribed** is owned, and hence identified, by the **Visit** of the **Patient**, and the **Lab-test**. Clustering algorithms that instruct the user to embed weak entity types within their owners, fall short of handling multiple owners, as demonstrated in this figure. Another complex situation involving weak entity types is that of cyclic owner dependencies. Ownership cycles are problematic for clustering instructions as above, and cannot be resolved by embedding all entity types in the cycle within their owner entity types. But the weak entity types in a ownership cycle can be resolved into entity types having *reference attributes* to each other.

An EER schema that includes weak entity types cannot be fully formalized, since the user must provide information about the intended status of weak entity types, i.e., whether a weak entity type symbol stands for a real entity type or for an attribute of another entity type. When the missing information is provided it can still be illegal since it might yield an illegal schema, or an inconsistent one. In addition, the presence of complex owner dependencies, e.g., cycles, complicate the formal treatment as well.

### 3 Resolving the status of Weak Entity types

Weak entity type symbols in the schema are resolved either to stay as entity type symbols, or to turn into attribute symbols associated with other entity types. In the first case, the owner entity

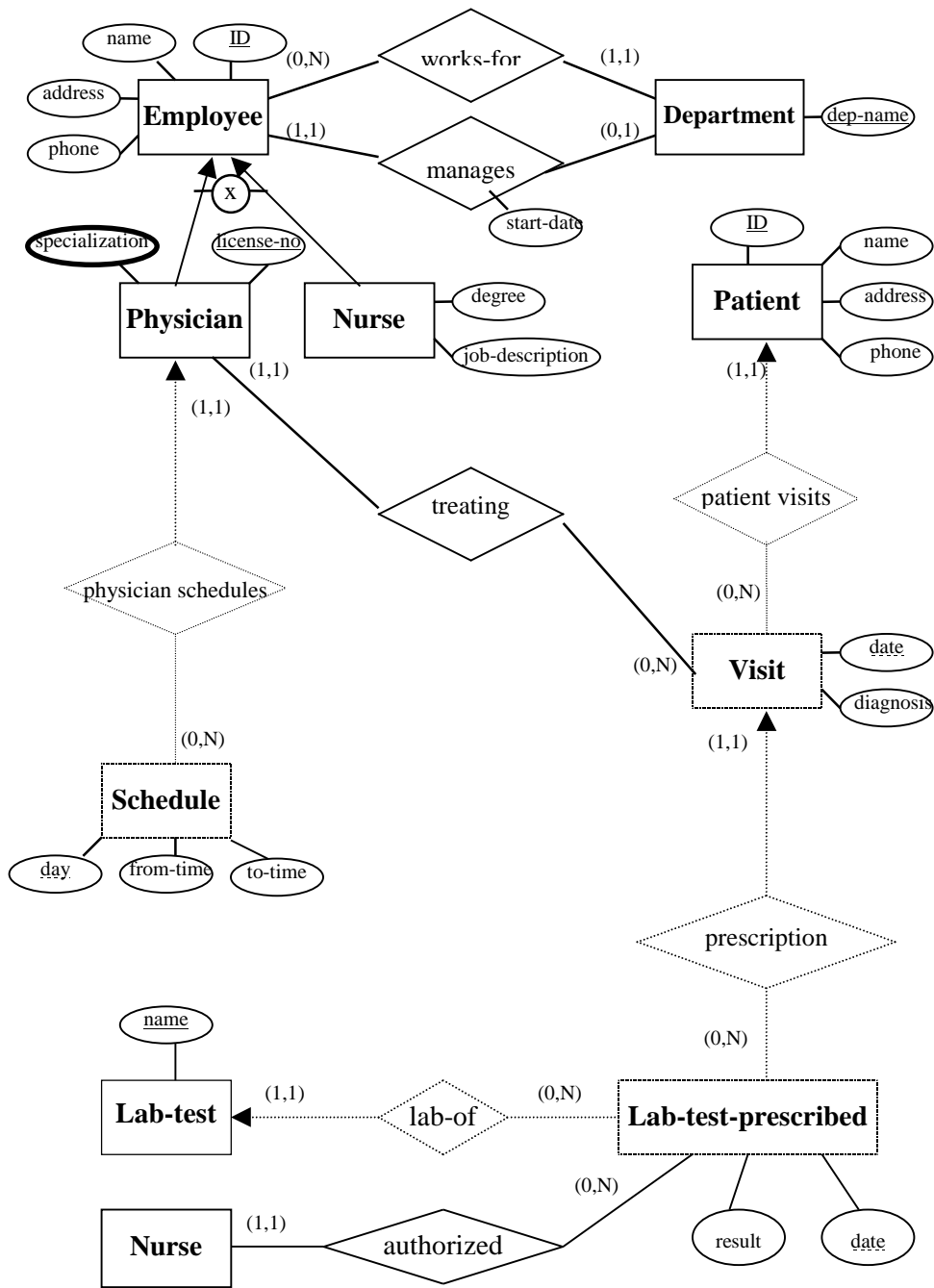


Figure 1: An EER diagram for a medical clinic information system

type symbols of the weak entity type symbol  $E$  are combined to form a new composite *reference attribute* attribute of  $E$ . In the second case, the weak entity type symbol becomes an attribute of its “closest” (direct or indirect) owner entity type symbol that is resolved as an entity type. In order to determine this closest entity type symbol, an inductive construction of the “target” entity type symbol is required.

The algorithm for removing the weak entity type symbols from an EER schema by resolving their final status works in three stages: In the first stage, the user marks, following some guidelines and constraints, the final status of the weak entity type symbols. In the second stage, the “target” entity type symbol for weak entity types that have an “attribute” destination is determined. In the third stage, the new EER schema, that includes no weak entity type symbol, is generated. The algorithm is introduced below, and its application to the medical clinic example from Figure 1 is demonstrated.

### **Stage I: Determining the final status of the weak entity type symbols:**

1. A weak entity type symbol with several owners, i.e., several identifying relationship types, is resolved to stay as an entity type. The rationale here is to prevent replication of the weak entity type as attributes of all of its owners. For example, in Figure 1, **Lab-test-prescribed** is resolved to stay as a regular entity type because it is owned by **Visit** and **Lab-test**.
2. A weak entity type symbol  $E$  with a single owner, such that  $E$  is not associated with a non-identifying relationship type symbol, and is not the owner of another entity type symbol, is resolved to be embedded as an attribute of its owner entity type symbol. For example, in Figure 1, **Schedule** is resolved to be embedded as an attribute of **Physician**.
3. Otherwise, the final status is determined by the user (modeler). That is, a weak entity type symbol that has a single owner, and is either associated with a non-identifying relationship type symbol, or is the owner of another weak entity type, is resolved either to stay as an entity type, or to be embedded as an attribute of its owner entity type symbol. For example, **Visit** in Figure 1, might have either an “attribute” or an “entity type” resolution. It is advised that if the weak entity type is constrained by a non-trivial (not  $(o, N)$ ) cardinality constraint, it is resolved to stay as an entity type.

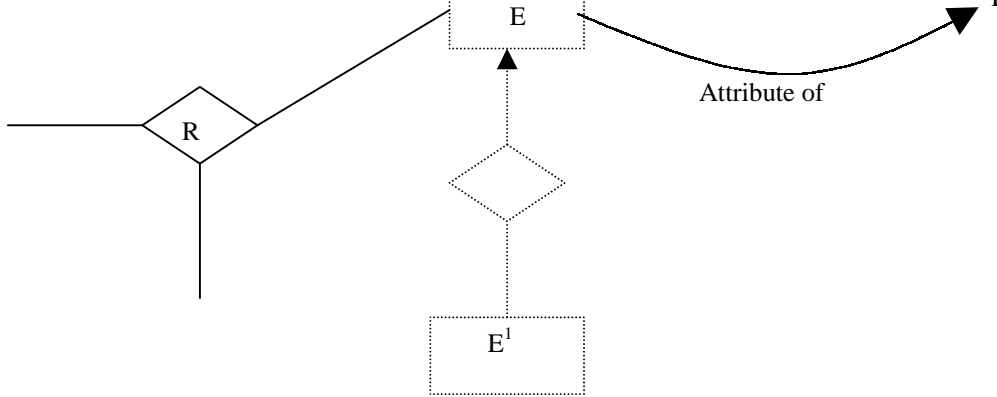


Figure 2: A weak entity type symbol with an “attribute” labelling, that is the owner of another weak entity type symbol, and is related through a non-identifying relationship symbol

**Stage II: Determining the “target” entity type symbol for weak entity types that have an “attribute” destination:**

The **target\_entity\_type** of  $E$  is the weak entity type symbol in which  $E$  will be embedded as an attribute. It is needed for the case where  $E$  is the owner of another weak entity type symbol  $E^1$ , that is resolved to stay as an entity type, or if  $E$  is related with a non-identifying relationship type symbol  $R$ . This situation is demonstrated in Figure 2. The entity type symbol  $E^1$  needs a *reference attribute* to  $E$ , as part of its key, and  $R$  needs to relate  $E$ , with other entity type symbols. Since  $E$  is resolved to turn into an attribute,  $E^1$  and  $R$  need a reference to the entity type symbol in which  $E$  is embedded. This is the **target\_entity\_type** of  $E$ . For example, in Figure 1, if **Visit** is resolved to be embedded as an attribute of **Patient**, then **treating**, needs to reference **Patient** as the **target\_entity\_type** of **Visit**. Similarly, **Lab-test-prescribed** which has an “entity type” resolution, needs **Patient** as a reference attribute.

1. For every entity type symbol  $E$  that is either strong or is resolved to stay as an entity type symbol, **target\_entity\_type**( $E$ ) =  $E$ .
2. For every entity type symbol  $E$  that is resolved to be embedded as an attribute, and is owned by the (single) entity type symbol  $E'$ : **target\_entity\_type**( $E$ ) = **target\_entity\_type**(  $E'$  ).

The **target\_entity\_type** labelling of entity type symbols in a schema can be computed inductively, starting from the strong entity type symbols, and the weak entity type symbols that have an entity type resolution. If at the end of this computation, there are entity type symbols without **target\_entity\_type** labelling, then the status decisions, taken in stage I are illegal, since there is a

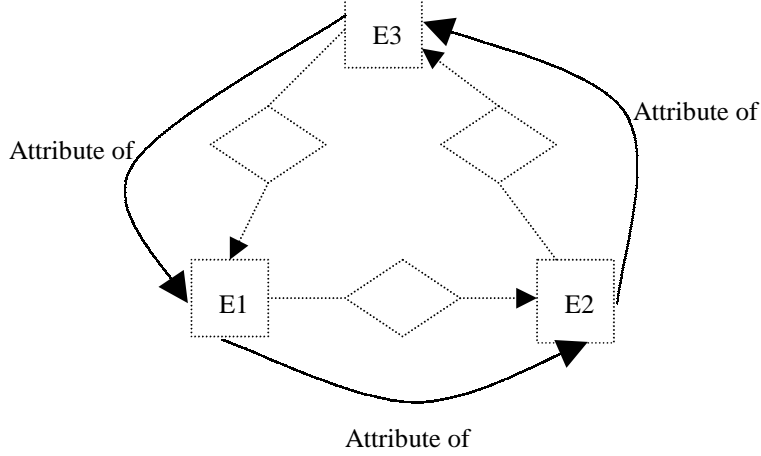


Figure 3: A cycle of owner entity type symbols

cycle of owner entity type symbols, all with attribute embedding decisions. This situation is demonstrated in Figure 3: The weak entity type symbols in the cycle cannot have a **target\_entity\_type** labelling since they cannot be reached by such an inductive construction. In that case, either output the entity type symbols without labelling and stop, or modify stage I.

The computation of the target entity type of an entity type symbol  $E$  in a schema, is associated with the cardinality constraints between  $E$  to its target entity type. This computation is obtained by composing the cardinality constraints along the role path between  $E$  to **target\_entity\_type**( $E$ ). For the sake of simplicity we avoid specifying this computation here.

**Stage III: Constructing a new EER schema  $\mathcal{ER}'$  that does not include weak entity type symbols, from a given schema  $\mathcal{ER}$ :**

Insert to  $\mathcal{ER}'$  all elements of  $\mathcal{ER}$ .

1. **Weak entity type symbols that have an “entity type” resolution:** For every such entity type symbol  $E$  that is owned by the entity type symbols  $E_1, \dots, E_k$ , with **target\_entity\_type**( $E_i$ ) =  $E_i^1$ ,  $i = 1, k$ , add to  $E$  a composite attribute named *owner* which is associated with the *reference attributes*  $E_i^1$ ,  $i = 1, k$ . Add *owner* to all keys of  $E$ . This situation is demonstrated in Figure 4. The newly added reference attributes are denoted with bold dashed circles. In Figure 7 **Lab-test-prescribed** has two reference attributes, **Lab-test** and **Patient**. If  $\mathcal{ER}'$  includes an entity type symbol with a reference attribute to itself, then the status decisions taken in stage I are illegal, since there is a cycle of owner dependencies: An entity type symbol  $E$  in  $\mathcal{ER}$  is owned by itself.

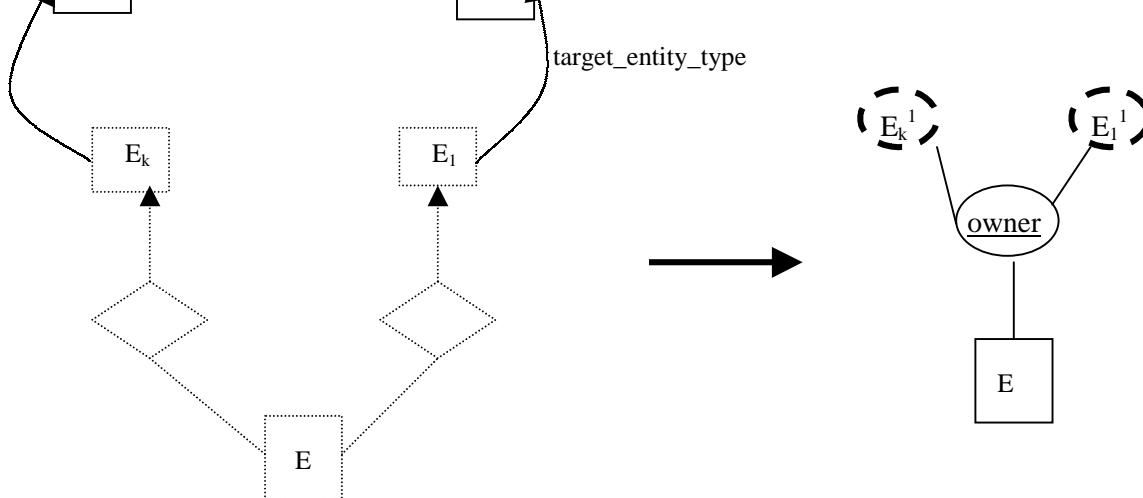


Figure 4: Reference attributes added to the key of a weak entity type symbol with an “entity type” labelling

For each identifying relationship between  $E$  to some  $E_i$  it is up to the user to decide whether it should be removed or not. Removing such a relationship implies that the user thinks that there is no need to relate entities of  $E_i^1$  to entities of  $E$ . The other direction is already covered by the newly added reference attributes of  $E$ . In any case, a relationship that is resolved to stay in  $\mathcal{ER}'$  is not identifying any more. In the medical clinic example of Figure 1, the decision was not to remove such relationships. Hence, in Figure 7 the relationships **Lab-of** and **prescription** stay as regular relationships.

2. **Non-identifying relationship type symbols:** For every relationship type symbol  $R$  that relates in  $\mathcal{ER}$  the entity type symbols  $E_1, \dots, E_k$ , replace them in  $\mathcal{ER}'$  with **target\_entity\_type**  $\_type(E_i) = E_i^1$ ,  $i = 1, k$ , respectively. Whenever  $E_i^1$  is different from  $E_i$ , the cardinality constraint on its role in  $R$  is set to  $(0, N)$ . This situation is demonstrated in Figure 5. After this stage, non-identifying relationship type symbols in  $\mathcal{ER}'$  relate only entity type symbols with “entity type” resolution.

3. **Weak entity type symbols that have an “attribute” resolution:**

While there exists in  $\mathcal{ER}'$  an entity type symbol  $E$  with “attribute” resolution, which is not the owner of another (weak) entity type symbol with an attribute resolution, repeat the following:

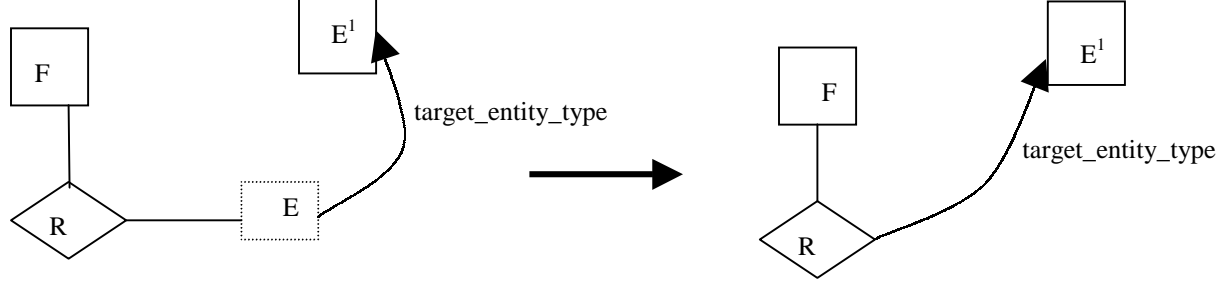


Figure 5: Transforming the reference in non-identifying relationship type symbols

- Assume that the owner of  $E$  in  $\mathcal{ER}'$  is  $E^1$ , via the identifying relationship  $R$ , and with role names  $RN$  and  $RN'$ , respectively. Assume that  $E$  has the attributes  $A_1, \dots, A_n$ . Then, add to  $E^1$  a composite attribute  $E$ , that is associated with the attributes  $A_{1\_of\_E}, \dots, A_{n\_of\_E}$ . If the cardinality constraint imposed on the role name  $RN$  is  $(1, 1)$ , i.e., at most a single  $E$  entity is identified by a given  $E^1$  entity, then the attribute  $E$  of  $E^1$  is marked as single-valued. Otherwise, it is multi-valued. This situation is demonstrated in Figure 6.
- Remove  $E$  and its identifying relationship from  $\mathcal{ER}'$ .

The schema that results from the application of this procedure to Figure 1 is described in Figure 7. In the new schema **Schedule** turns into an attribute of **Physician** since it has a mandatory attribute labelling, according to Stage I. **Lab-test-prescribed** stays as an entity type symbols since it has two owners in the original schema. It has two reference attributes, **Lab-test** and **Patient**, which are the **target\_entity\_types** of **Lab-test** and **Visit**, respectively. The decision to turn **Visit** into an attribute of its **target\_entity\_type Patient**, was taken mainly in order to demonstrate the case of “attribute” labelling for a weak entity type symbol that is related through a regular relationship type, and is also the owner of another weak entity type symbol with an “entity type” labelling.

**Modification of the EER datamodel definition:** Two modifications are needed in the schema definition and its semantics:

1. **Reference attributes:** This is a special kind of attribute symbols, that take the same name as an entity type symbol in the schema. The type of a reference attribute symbol  $E$  of a

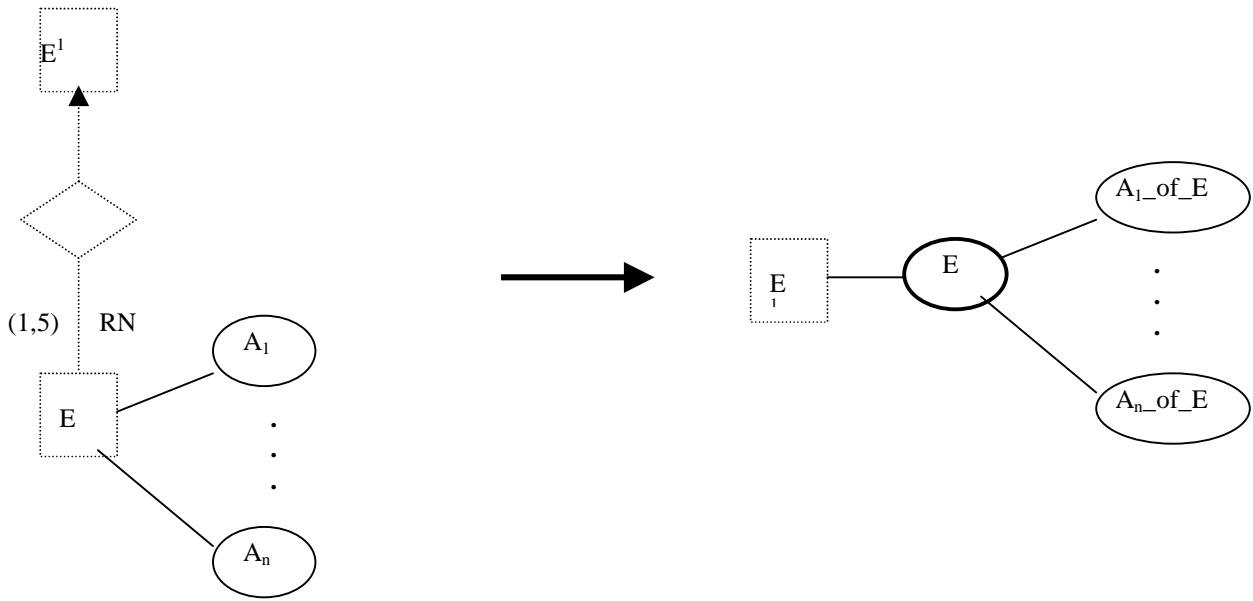


Figure 6: Embedding a weak entity type with an “attribute” resolution as an attribute of its owner

type symbol  $T$  must be included in the domain of entities. If  $dom(E, T)$  denotes the type assigned to the attribute  $E$  of  $T$ , and  $D$  is the domain of entities in a database instance of the schema, then  $dom(E, T) \subseteq D$ .

2. **Consistency:** Additional condition for the consistency of a database instance  $\mathcal{D}$  of a schema: For every reference attribute symbol  $E$  of a type symbol  $T$ , its type is exactly the extent of  $E$  in  $\mathcal{D}$ . That is,  $dom(E, T) = E^{\mathcal{D}}$ .

## 4 Conclusion

This work is part of our more general work on formalizing the EER model for database programming level. We encountered problems with handling weak entity type symbols in two cases. First, in our work on the MEER model, which is an EER schema that is enhanced with structure methods that are cardinality sensitive ([1]). In order to implement the MEER model on top of a database schema (as in [2]), there is a need to implement all the embeddings of weak entity type symbols, and to develop the machinery to reference these embeddings. This process becomes unreasonably complex, and calls for separation between the status resolution of the weak entity type symbols,

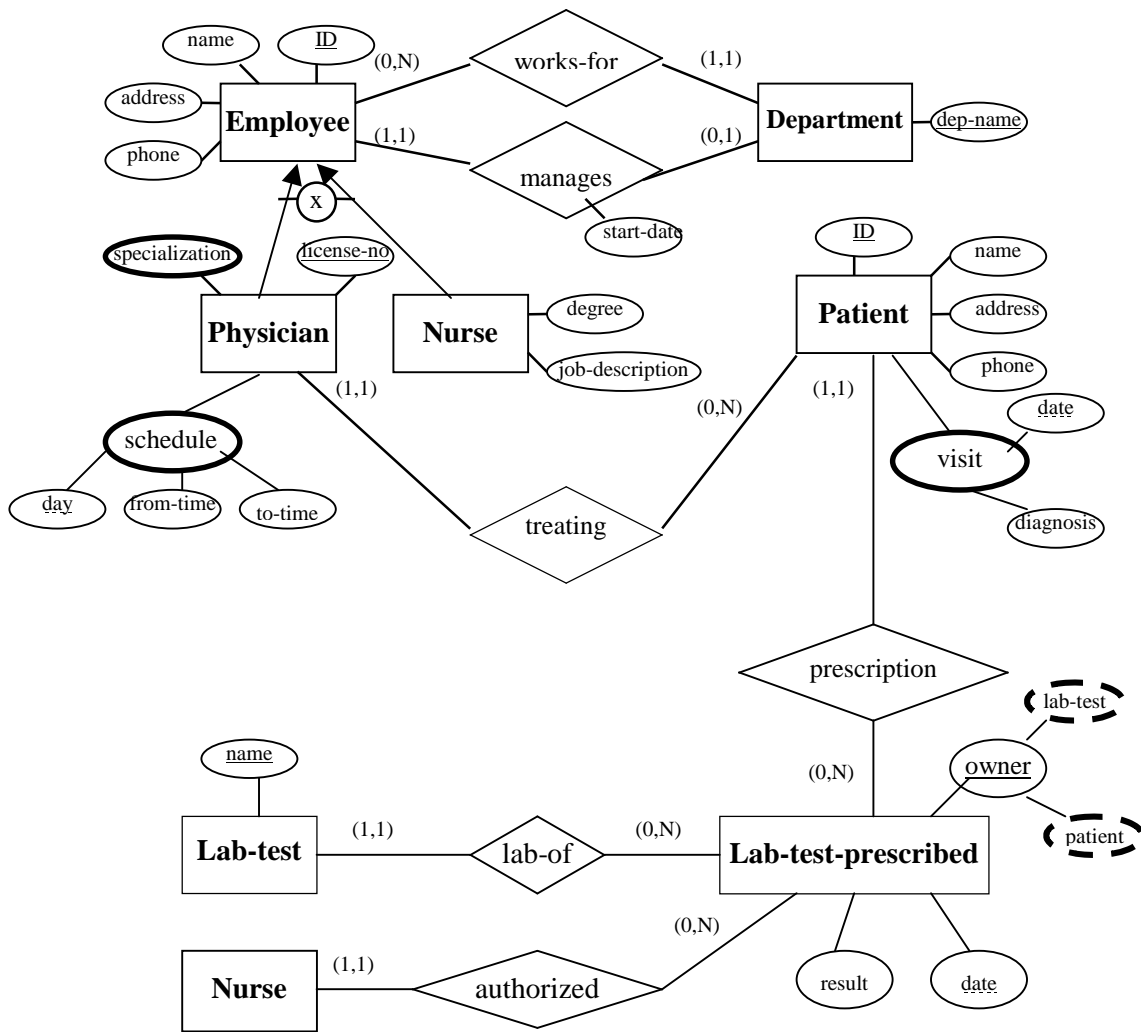


Figure 7: A Weak-entity-type-symbols less EER diagram for the medical clinic information system

to the implementation of the methods. Once the two processes are separated, it makes sense to abstract the weak entities part on the EER conceptual level. This way we separate this process from the implementing database model. The other case is in our work on abstraction, hierarchy and clustering in EER diagrams. Most suggestions try to cluster weak entities with their owners. But this process is not well-defined due to the complex structure of weak-owner dependencies, as described in this paper.

In this paper we introduced an algorithm for resolving the status of weak entity types in EER schemas. If the algorithm succeeds, it yields an EER schema without weak entity types. The contribution of this paper is in resolving, once and for all, the weak status of weak entity types in EER schemas. The algorithm we introduce can be applied as the last step in the design of the schema. Once this is done, all existing formal investigations of EER schemas are applicable, and EER schemas can be formally defined as an abstract conceptual level on top of a logical database schema.

In the Object-Oriented (OO) data model there are no “weak classes”, since there is no mandatory key requirement. Yet, the conceptual role played by weak entity types seems useful, since it enables to mark classes that might, later on be embedded as attributes of other classes. The algorithm for resolving the status of weak entity types applies in this case as well.

## References

- [1] M. Balaban and P. Shoval. Enhancing the ER model with structure methods. In *CAiSE'98/IFIP Workshop on Evaluation of Modeling Methods in Systems Analysis and Design*, June 1998.
- [2] M. Balaban and P. Shoval. Enhancing the ER model with structure methods. *Journal of Database Management*, To appear, 1999.
- [3] M. Balaban and P. Shoval. EER as an active conceptual schema on top of a database schema – object-oriented as an example. Technical report, Information Systems Program, Department of industrial Engineering and Management, Ben-Gurion University of the Negev, ISRAEL, March 1999.

- [4] D. Calvanese, M. Lenzerini, and D. Nardi. Description logics for conceptual data modeling. In J. Chomicki and G. Saake, editors, *Logics for Databases and Information Systems*. Kluwer Academic Publishers, 1998.
- [5] P.P. Chen. The entity-relationship model: toward a unified view of data. *ACM Transactions on Database Systems*, 1(1):9–36, 1976.
- [6] R. Elmasri and S.B. Navathe. *Fundamentals of Database Systems*. The Benjamin/Cummings Publishing Company, 1994.
- [7] P. Jaeschke, A. Oberweis, and W. Stucky. Extending er model clustering by relationship clustering. In *Entity-Relationship Approach*, pages 451–462. North-Holland, 1993.
- [8] M. Lenzerini and P. Nobili. On the satisfiability of dependency constraints in entity-relationship schemata. *Information Systems*, 15(4):453–461, 1990.
- [9] B. Thalheim. *Fundamentals of Entity-Relationship Modeling*. Springer-Verlag, 1988.
- [10] B. Thalheim. Fundamentals of cardinality constraints. In *Entity-Relationship Approach*, pages 7–23. North-Holland, 1992.