A survey of database design transformations based on the Entity–Relationship model

Christian Fahrner, Gottfried Vossen*

Institut für Wirtschaftsinformatik, Universität Münster, Grevenerstr. 91, D-48159 Münster, Germany

Received 22 August 1994; revised manuscript received 1 October 1994; accepted 21 February 1995

Abstract

At present, the Entity–Relationship (ER) model is the most important paradigm for conceptual database design. Since the model was introduced in the mid-seventies, a large body of literature has been published on transforming conceptual ER schemas or diagrams into logical data models. The purpose of this paper is to survey this literature. A first focus is on transformation approaches from the ER model to traditional data models, i.e. to the relational, the network, or the hierarchical model; this is then complemented by a discussion of more recent transformations to object-oriented models. A second focus is on considering the process of reverse engineering, i.e. transformations from a logical model back into the ER model; finally, an overview of direct transformations between various logical data models is presented.

Keywords: Conceptual database design; Entity–Relationship modeling; Schema transformations; Reverse engineering

1. Introduction

The Entity–Relationship (ER) model, originally proposed by Chen in 1976 [22], has gained wide acceptance in the area of database design and related fields. A reason for this is the simplicity and clarity of its structuring concepts, which allow users to model real-world objects and their relationships in a natural way. In database design, it is commonly employed in the conceptual design phase, whose goal is to obtain a system-independent global view of a given application for which a database is under construction. Once an ER diagram, the result of conceptual design, has been obtained, a database designer has to transform that into the logical data model of a target system. Numerous approaches have been proposed in the past for accomplishing this task; the goal of this paper is to survey such approaches.

* Corresponding author. Email: vossen@uni-muenster.de

1 This work has been sponsored by the Deutsche Forschungsgemeinschaft under Grant Vo 426/7–1.
Concepts for transforming an ER schema into another data model go back to the work of Chen [22], who showed that the ER model allows for easy mapping into traditional data models, including the relational, the network, and the hierarchical model. This has made the ER model the most popular tool for conceptual database design; at present, several design tools centering around the ER model are even commercially available [88], and others are under development [4]. Another discovery that was made soon after the introduction of the model was that the ER formalism actually provides a conceptual framework in which a number of design aspects can be captured. This led to numerous extensions of the original model and, consequently, to numerous proposals for transforming such an extension into a commercial data model. As a result, the literature in this area is now huge, and it has become difficult for a database designer to pick the appropriate version of the ER model together with the relevant transformation techniques. For example, distinct versions of the model use constructs which are equally named, but have a different formal definition as well as a different semantics. Also, a large number of graphical symbols for denoting ER constructs has been proposed, which can even be confusing for a database designer.

In this paper, we survey major portions of the relevant literature in this field. We do so by making our description of the ER models in use as general as possible for reasons of understandability. Our first emphasis is on so-called forward engineering transformations, which are the classical ones used in database design. Indeed, we survey a number of distinct approaches for converting an ER diagram into a database schema of one of the traditional data models, where an emphasis is put on transformations into the relational model; transformations into the hierarchical or network model are only sketched for completeness reasons. While database systems based on the latter two models are of decreasing importance today, there is an emerging interest in using ER techniques for designing object-oriented databases. Therefore, we also describe ways of transforming from ER to an object-oriented (OO) data model. However, as will be seen, most relevant approaches proposed so far are tailored towards a specific OO system, which can be explained by the fact that a common standard in the area, the ODMG-93 proposal [21], has only recently been published, and its commercial availability is expected only for mid-1995.

A second emphasis of our survey is on the converse process of reverse engineering, i.e. transformations from some logical model back into the formalism of the ER model. Reverse engineering is also of increasing importance nowadays, for example in the context of using legacy systems together with novel types of database systems, or of migrating from traditional to more recent systems. Specifically, we survey reverse engineering techniques centering around traditional as well as object-oriented data models, which is central in this context, but clearly not the whole story (indeed, reverse engineering ideally also takes information provided by existing application programs or at least SQL queries [104] into account, which is not considered here). Finally, we even consider ways to convert a schema in one of these models directly into a schema in another such model, without going back to an ER description. As for forward engineering transformations, we here focus again on heterogeneous transformations involving the relational and object-oriented models for the same reasons as above. On the other hand, it is worth mentioning that direct transformations of a network or hierarchical schema into an object-oriented one do not seem to exist.

An overview of the organization of this paper is shown in Fig. 1, where arrow labels indicate
relevant sections. In detail, the organization of this paper is as follows: Section 2 is a brief introduction to the database design process and to the various directions in which transformations can occur in that process. Section 3 establishes characteristic properties according to which different transformations can be evaluated; this provides a kind of framework into which the approaches discussed in this paper will be cast. In Section 4 we discuss forward engineering transformations, in particular transformations from the ER model to a logical data model. In Section 5 we discuss approaches to do the reverse, and in Section 6 we discuss approaches to transform database schemata directly from one logical model to another. We mention that our exposition in Sections 4–6 is incomplete in that we concentrate on describing approaches to forward and reverse engineering which we consider significant, but there is still a number of additional papers making minor contributions to the subject. Finally, Section 7 presents several conclusions we have obtained from this study.

2. Database design

The general task of database design is to map a given real-world application into the formal data model of a given database management system. We assume the reader to have a general understanding of database design, for example along the lines of Batini et al. [5], Elmasri and Navathe [36], or Vossen [105]. In this section, we briefly clarify the various steps that have to be performed when doing database design. In particular, we consider the important step of converting a conceptual description of a given application into a logical one; in addition, we briefly discuss transformations that deal with the converse direction. Finally, we indicate why direct mappings between logical models are of interest as well.

Following Batini et al. [5], database design can roughly be seen as a four-step process as shown in Fig. 2. The input to a database design process commonly is a variety of information in various forms, e.g. textual, formal, or semi-formal, which may arise from interviews with potential users, former design processes, previously designed databases, etc. The goal is to produce from that a database schema which can be processed by a database management system. In the first step, requirements analysis, all information needed to correctly model the desired application is collected and analyzed. The primary goal is to collect data requirements, i.e. what the user wants to store data about, how this might be structured, and how it might be interrelated. Correspondingly, the input of this step are data requirements in various forms,
Requirements analysis

Analysed Requirements

Conceptual design

Logical design

Logical schema

Physical design

Physical schema

Application programs

Data and application requirements

Fig. 2. The database design process.

e.g. natural language, forms, record formats, or views. The output are requirements in some structured form, which represent a first abstraction of the underlying application.

In the second step, a conceptual schema is created, describing the structure of the database independent of the underlying system. A conceptual model is used to describe the conceptual schema. Most common today is the use of the Entity–Relationship (ER) model in this step, but other semantic models might be appropriate as well. In the third step, the conceptual schema is transformed into a logical schema which is specified by a logical model. A logical schema is a description of the structure of the database that can be processed by a database system. In practical applications, the relational model is still the most important logical model, although there is an increasing interest in higher logical models, in particular object-oriented ones. In the fourth and final step, a physical schema is derived from the logical one, which roughly describes the implementation of the database in secondary storage. Storage structures and access paths for database files are specified to achieve good performance for the given application. This step depends on the underlying system and on the transactions described for the database.

In parallel to this data-driven design, a function-driven design [5] can be performed to model operations and transactions that should be processed on the database. This design
process starts with the collection of processing requirements and ends with a specification of application programs. Various techniques can be used to support this design including HIPO (Hierarchical Input Process Output), SADT (Structured Analysis and Design), DFDs (Data Flow Diagrams) or Petri nets; details are omitted. Clearly, data design and functional design are strongly connected with each other. A detailed description of both and their relationship can be found, for example, in Batini et al. [5] or Elmasri and Navathe [36].

In this survey, we concentrate on Step 3, which is often called a *forward engineering transformation*. In addition to forward design, much interest exists nowadays in the opposite process, commonly subsumed under the term *reverse engineering*. Here, the starting point is an existing database description, and the goal is to obtain a conceptual description of the application under consideration. The motivations for reverse engineering are manifold, e.g. to re-use existing database designs, to migrate from one database system to another, or to get a conceptual view of the application modeled in a database. However, reverse engineering in databases is not just a simple inversion of previously performed forward engineering, but it is a process which tries to elicit the semantics of an application from the abstractions modeled in objects and their relationships with the help of a semantic data model.

Since the interest in doing reverse engineering on databases is still pretty new, there is not yet an accepted strategy for performing it. Indeed, only a few publications investigate reverse engineering techniques which are applicable to different (source) models; for example, Hainaut et al. [44, 45, 46] propose a general, two-step procedure for reverse engineering (cf. Fig. 3): In the first step — *Data Structure Extraction* — a complete description of the source schema is produced by eliciting information given only implicitly in the database description, e.g. in the contents of the database, in application programs, or in views. This step can even be seen as the inverse of physical design. In the second step — *Data Structure Conceptualization* — the structures of the logical schema are first analyzed and restructured and then transformed into structures of the semantic target model. This step can be seen as an inversion of logical design.

![Diagram of reverse engineering process](image)

*Fig. 3. Database reverse engineering (cf. Hainaut et al.)*
An important difference between forward and reverse engineering is that the latter typically requires user interaction, while the former can vastly be automated. There are various reasons for this, including the following [45, 46, 85, 86]:

- Information about the structure and semantics of a source schema is often not given explicitly in the schema description, but implicitly in application programs. The semantics of the latter is difficult to extract for a program.
- The modeling constructs of the source data model are often semantically overloaded with respect to the constructs of the semantic target model. This means that given structures in the source schema can often be interpreted as and transformed into different structures of the target model.
- The structures of the source schema could have been optimized for performance reasons. Therefore, relationship structures might be difficult to discover and to interpret.

As a consequence, reverse engineering often requires some form of semantic enrichment of a given database schema (called data structure extraction above); see also [19, 39]. Relevant approaches will be discussed in more detail in Section 5.

Besides the two transformation directions we have discussed so far, in top-down direction for forward or bottom-up direction for reverse engineering, a third approach to transforming data models is to operate directly at the level of (distinct) logical models. Transformations of this kind are confronted with similar problems as those encountered in forward or reverse engineering. Their justification is that structural aspects of the traditional models have many commonalities, so that it often causes an unacceptable additional effort to take the ER-model as intermediate model, which requires semantic enrichment, instead of mapping directly from one logical model to another.

3. Properties of transformations

Before we embark on a detailed description of transformations used in conceptual and logical database design, we gather a number of properties by which such transformations can be evaluated and compared. Indeed, each approach for one of the transformation directions mentioned earlier has its own characteristics and properties, e.g. including necessary preconditions or the transformation methodology employed. We now survey those properties that are most relevant to our exposition in the following sections.

- **ER model used:** As we mentioned in the Introduction, numerous extensions to the modeling structures of the original ER model of Chen [22] have meanwhile been proposed, including the ECR-model [36], the ECR+-model [94], HERM [100], or the EER-model [99]. They mainly differ in the structures that are supported, e.g. generalization structures, complex attributes, aggregations, and relationships over relationships. Formal definitions of these constructs, even if they are equally named, typically vary from one model to the next; on the other hand, it turns out that the conceptual semantics of equally named constructs is often the same.
- **Transformation prerequisites:** Some algorithms require the source schema to be in a special
form, e.g. normalized in some way, or to satisfy specific conditions, e.g. unique naming of attributes. Sometimes a set of integrity constraints is required to be given as a prerequisite; for example, relational reverse engineering approaches are often based on a complete set of inclusion dependencies, keys, or foreign key constraints.

- **Integrity constraints:** A schema description typically contains inherent, implicit and explicit integrity constraints [36]. Some transformation approaches take all of these into account, while others consider them partially only, or ignore them completely.

- **User involvement:** In contrast to forward engineering, reverse engineering and heterogeneous transformations between logical models often require user interaction. The amount of user interaction is dependent on the kind of heuristics that is applied and on the assumptions made.

- **Operations:** Taking the operations that should be processed on a database over the target schema into account during a translation can improve performance. Considering processing requirements is particularly important for models in which logical and physical design is difficult to separate, like the hierarchical and the network model. In the relational model, optimization with respect to operations is often performed in an extra step after the transformation, i.e. in the phase of physical design. This aspect is only of interest for transformations into a logical model.

- **Information preservation:** The goal (and at the same time difficulty) of any schema transformation is to convert a schema in a source data model into a schema with the same information capacity [5, 48, 50, 65, 73, 102] of the target model. Formally, this means that all possible database instances which can be represented in the source schema can also be represented in the target database schema and vice versa. If a transformation satisfies this property, it is called information preserving. If modeling constructs of high semantic abstraction cannot be represented directly, or be simulated through structures and integrity constraints of the target model, it becomes difficult to satisfy this property; this is even made impossible if structural information and integrity constraints are buried in application programs. As will be seen, only a few approaches take the problem of correctness into account.

- **Quality of the resulting schema:** There are several schema quality criteria which can even be defined for schemas in different models. The three most important ones for the ER model [5] are correctness, minimality, and normality. A schema is **correct** if all concepts of the underlying model are used correctly with respect to syntax and semantics. A schema is **minimal** if all redundancies have been removed from it, and it is **normalized** if it satisfies one of the well-known normal forms. Other criteria include readability, self-explanation, and extensibility. Clearly these quality criteria can also be defined for schemas in other models. It must be mentioned that the quality issue of a transformation result is often neglected, especially in reverse transformations.

- **Transformation methodology:** Each transformation typically has its distinct methodology. However, transformations for a specific direction can sometimes be classified with respect to a common methodology; this will be worked out as far as possible in the remaining sections.

In the following sections the different approaches will be evaluated with respect to these points whenever suitable or applicable.
4. Forward engineering: From ER to a logical model

In the following it is assumed that the reader is familiar with the basic concepts of the ER, network, relational, and hierarchical data models. Descriptions of their basic concepts can be found in many database textbooks, e.g. [5, 36, 61, 63, 103, 105]. Descriptions of the ER extensions ECR, EER and ECR + can be found in [37], [99, 101] and [94]. For simplicity, we will often use the terms relation, entity, relationship, record, and set instead of relation schema, entity type, relationship type, record type, and set type, in particular if the meaning of these terms is clear from the context.

The major difficulty of logical database design, i.e. of transforming an ER schema into a schema in the language of some logical model, is the information preservation issue. Indeed, assuring a complete mapping of all modeling constructs and constraints which are inherent, implicit or explicit in the source schema is problematic since constraints of the source model often cannot be represented directly in terms of structures and constraints of the target model. In such cases, they must be realized through application programs; alternatively, an information-reducing transformation must be accepted.

4.1. From ER to relational

There is a large body of work devoted to the transformation of ER schemas into relational schemas. The basic ideas of such a transformation are described in most database textbooks, including [5, 36, 61, 63, 68, 103, 105]. Other publications investigating in this transformation direction are, for example, [1, 7, 13, 12, 22, 30, 33, 34, 55, 62, 66, 69, 70, 71, 72, 73, 82, 87, 95, 97, 101, 107]. Many of these focus on the original version of the ER model [13, 22, 30, 34, 55, 70, 82, 103, 107]; others employ an extended ER model, including subtype/supertype relationships or more complex generalization structures [5, 7, 12, 33, 36, 62, 63, 66, 68, 95, 101, 105]. Only a few are based on an extended ER model with aggregation abstraction or relationships over relationships [1, 61, 69, 71–73, 97]. What all these approaches have in common is that they do not need user interaction and, with the exception of [7], do not consider operational issues. The approaches just mentioned are based on the same basic methodology, which we hence call the general transformation approach. We describe this approach first and then turn to specific issues going beyond it.

4.1.1 The general transformation approach

The general transformation approach for ER structures is based in the following types of steps:

- Each entity type is translated into a relation. Attributes and the primary key of the type become attributes and the primary key of the relation.
- Weak entity types are translated into a relation. Beside the local attributes, the primary key attributes of the identifying entity type are included as attributes. The primary key of the relation consists of the local key attributes plus the primary key attributes of the identifying entity type.

In general, relationship types are transformed based on their arity and on the cardinality constraints required for their components as described below, where IS-A relationship types
and generalizations are treated separately from the other relationship types. Let us first consider the most important case of binary relationship types, as shown in Fig. 4; we assume that the cardinality constraints are given as Min-Max constraints as described, for example, in [36]. Binary relationship types can be transformed in three different ways, depending on the cardinality constraints specified for their components (cf. Fig. 4):

**Option 1:** Definition of a separate relation. The primary key attributes of the relationship components A and B and the attributes of the relationship R are included as attributes in a newly created relation \( Rel_R \). If the Max cardinality of a component equals 1, the primary key of the component can be chosen as primary key of \( Rel_R \); otherwise, the union of the primary keys of A and B are chosen as key. This transformation can be applied to every kind of binary relationship type, independent of the cardinality constraints. Beside the mandatory usage of this option if both components have a Max cardinality greater than 1, it is also often used in practice if the relationship type has attributes of its own or if both components have an optional participation (i.e. Min cardinalities equal 0). Under this option, the transformation of relationship R from Fig. 4 results in the following three relations, where primary keys are underlined:

\[
Rel_A(K_1,\ldots) \quad Rel_R(K_1,K_2,\ldots) \quad Rel_B(K_1,\ldots)
\]

**Option 2:** Insertion of a foreign key. The primary key of the relation corresponding to one component, say B, is inserted as foreign key in the relation corresponding to the other component A. If relationship attributes exist, these are also inserted into A. This option can only be applied if the Max cardinality of A equals 1. If both components have a Max cardinality of 1, the foreign key is included in that relation whose component has a mandatory participation in R (i.e. its Min cardinality equals 1). Note that mandatory participation may be required to avoid null values for foreign keys. For the example from Fig. 4, this option leads to the following two relations:

\[
Rel_A(K_1,K_2,\ldots) \quad Rel_B(K_2,\ldots)
\]

**Option 3:** Collapsing A, B, and R into one relation. This option is only meaningful, if the Min-Max cardinalities c and d are both (1,1). In this case, the primary key of one of the components is chosen as primary key of the collapsed relation \( Rel_{A-B} \). For the example from Fig. 4, this results in the following relation:

\[
Rel_{A-B}(K_1,K_2,\ldots)
\]

The various possibilities to transform the binary relationship type R from Fig. 4 are summarized in Table 1 with respect to the cardinality constraints. The table does not take into account the presence of relationship attributes. If such attributes exist, some approaches prefer the creation of an individual relation for this relationship type, like Storey [97] or Markowitz [73], while others make no difference whether a relationship type has attributes or

---

**Fig. 4.** A binary relationship type R with components A and B.
Table 1
Various possibilities to represent binary relationship types

<table>
<thead>
<tr>
<th>Cardinality constraints</th>
<th>Transformation possibilities</th>
</tr>
</thead>
<tbody>
<tr>
<td>1:N-notation c:d</td>
<td>Option 1</td>
</tr>
<tr>
<td>Min-Max (c₁,c₂)-(d₁,d₂)</td>
<td>possible</td>
</tr>
<tr>
<td>1:1</td>
<td>x</td>
</tr>
<tr>
<td>(0,1)-(0,1)</td>
<td>x</td>
</tr>
<tr>
<td>(1,1)-(0,1)</td>
<td>x</td>
</tr>
<tr>
<td>N:1</td>
<td>x</td>
</tr>
<tr>
<td>(0,1)-(0/1,n)</td>
<td>x</td>
</tr>
<tr>
<td>(0/1,n)-(0/1,n)</td>
<td>x, x</td>
</tr>
<tr>
<td>N:M</td>
<td>x, x</td>
</tr>
</tbody>
</table>

not, like Teorey [101]. Approaches supporting only the 1:N notation for cardinality constraints without a distinction of optional and mandatory participation (see Column 1 in Table 1) usually use Option 2 above whenever applicable.

In Table 1, an ‘x’ in a column means that the corresponding option is applicable; an ‘x’ indicates that this option is commonly used in approaches supporting cardinality constraints with Min-Max notation or 1:N notation with mandatory/optional distinction, and an ‘x’ indicates common usage of the option for 1:N notation without mandatory/optional distinction.

We next consider the transformation of other relationship types which can occur in a given ER diagram:

- **Relationship types of arity > 2** are handled by creating a new relation having as primary key the union of the primary keys of the relationship components, and the attributes of the relationship type as additional attributes.

- **Recursive relationship types** are translated like non-recursive relationships, with the exception that role names must be provided for the key attributes representing the relationship.

- **Aggregations or relationship types which are a component of another relationship type** are transformed into an individual relation in the same way as relationship types with arity > 2.

- **IS-A relationship types** can be represented by making the key of the supertype a foreign key in the relation representing the subtype. Other possibilities which are often used for the representation of more complex generalization hierarchies are as follows:
  - The supertype S and its subtypes Sᵢ, 1 ≤ i ≤ n are represented together in a single relation Relₛ. The attributes of Relₛ are the union of the attributes of S and of the Sᵢ’s. Discriminating or role attributes can be added to indicate membership of some entity in a subtype Sᵢ; this transformation is meaningful if the subtype has no local attributes or at least no local key.
  - The supertype is not represented as individual relation, but its attributes are inserted into each relation representing a subtype. This transformation is only practical if the generalization hierarchy is total and exclusive.

Besides key constraints it is necessary to derive several other kinds of integrity constraints to obtain an information preserving transformation. These include foreign key constraints
(eventually in the form of key-based inclusion dependencies) to indicate the relationships between the relevant relations, existence dependencies (especially not-null constraints) to represent a mandatory participation of a component in a relationship type or the dependence of a relationship type on its components, and inclusion dependencies to represent a mandatory participation of components in a relationship in case this cannot be represented through existence constraints. Additionally, exclusion dependencies can become necessary to indicate exclusive generalization hierarchies; details are omitted.

Most approaches following this general methodology have only subtle differences. For example, Chen [22] and Jajodia et al. [55] transform every entity- and relationship type into an individual relation, which is also often called a canonical transformation. Dumpala and Arora [34], Teorey [101], or Elmasri and Navathe [36] support 1:N-cardinality constraints without distinguishing optional and mandatory participation and therefore always use the foreign-key representation for 1:N- and 1:1-relationships, while Batini et al. [5] and Storey [97] use all aforementioned possibilities with respect to Min-Max cardinalities. A major difference between all these approaches lies in the constraints derived during a transformation.

4.1.2 Other approaches

Approaches going beyond the general transformation approach can be found in [1, 5, 7, 33, 36, 66, 69, 68, 70–73, 95, 101]. These transformations are surveyed in Table 2 with respect to the following aspects:

- **ER model used**: Column 2 characterizes the extended ER model used in the respective approach. We distinguish simple subtype/supertype and IS-A relationships from more

<table>
<thead>
<tr>
<th>Approach</th>
<th>ER-model</th>
<th>Prerequisites</th>
<th>Derived constraints</th>
<th>Normalization</th>
<th>Inform. pres.</th>
<th>Canonical name ass. preproc.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Azar [1]</td>
<td>1,3,5,6,7</td>
<td>Y</td>
<td>K, FD, IND, JD</td>
<td>1NF</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>Batini [5]</td>
<td>1,2,5,6</td>
<td>N</td>
<td>S, F</td>
<td>K, FK</td>
<td>1NF</td>
<td>N</td>
</tr>
<tr>
<td>Bertaina [7]</td>
<td>1,2,5</td>
<td>N</td>
<td>K</td>
<td>1NF</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>Dogac [33]</td>
<td>1,2,3,4</td>
<td>N</td>
<td>Trigger</td>
<td>1NF</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>Elmasri [36]</td>
<td>1,2,4,6</td>
<td>Y</td>
<td>K, FK</td>
<td>1NF</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>Ling [66]</td>
<td>1,2,4,6,7</td>
<td>N</td>
<td>K, FD, FK</td>
<td>3NF/5NF</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>Mannila [68]</td>
<td>1,4,7</td>
<td>Y</td>
<td>K, IND</td>
<td>BCNF</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Markowitz [73]</td>
<td>1,3,4</td>
<td>Y</td>
<td>K, IND, NN</td>
<td>BCNF</td>
<td>Y</td>
<td>N</td>
</tr>
<tr>
<td>Springsteel [95]</td>
<td>1,4</td>
<td>N</td>
<td>K, IND</td>
<td>1NF/BCNF</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Storey [97]</td>
<td>1,2,3,5</td>
<td>N</td>
<td>K, IND</td>
<td>1NF</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>Teorey [101]</td>
<td>1,2,5</td>
<td>N</td>
<td>K, FK, NN</td>
<td>1NF</td>
<td>N</td>
<td>N</td>
</tr>
</tbody>
</table>

**Model**: 1 = Subtype/Supertype or IS-A relationships 2 = More complex generalization structures 3 = Aggregation or relationships over relationships 4 = 1:N-cardinality constraints 5 = Min-Max - cardinality constraints or 1:N-constraints with mandatory vs. optional distinction 6 = Complex attribute structures 7 = More than one key can be specified for each type

**ER preprocessing**: N = ER normalization S = Simplification of complex structures F = Flattening complex attribute structures

**Derived constraints**: K = Keys FD = FDs beyond keys IND = Inclusion dependencies NN = Not-Null constraints FK = Foreign key constraints JD = Join dependencies Y = Yes N = No
complex generalization structures, since the latter contain more semantic information, e.g. exclusion dependencies implied by exclusive generalization hierarchies. Therefore, it is more tedious for the latter to achieve information preservation. The same is true for the distinction of different kinds of cardinality constraints, in particular Min-Max constraints, 1:N-constraints with optional/mandatory distinction, or 1:N-constraints without this distinction.

- **Preprocessing of the ER schema:** Column 3 states whether some form of preprocessing is required for the initial ER schema, in particular normalization or simplification of constructs with high abstraction level to constructs with a lower abstraction level. In such a simplification process, generalization structures may get replaced by relationship types, or relationship types are made binary. A replacement of complex attribute structures by flat ones is both a normalization and a simplification; this is distinguished in the table. Complex attribute structures do not always require preprocessing, since sometimes flattening is done directly during the mapping.

  We also mention proper name assignment in Column 3, which is often ignored. The difficulties which can arise from improper naming w.r.t. to relational normalization are described in [71, 73].

- **Derived relational integrity constraints:** Column 4 lists relational integrity constraints which are derived during the respective transformation. Derived constraints can be keys, which are equivalent to key-based functional dependencies, functional dependencies (FDs) which are not key based, foreign-key constraints (FKs), inclusion dependencies (INDs), join dependencies (JDs), and a special form of existence constraints, the Not-Null-constraints (NNs). It is worth noting that FKs are a special form of inclusion dependencies, and that most approaches derive only INDs indicating a foreign-key constraint; since INDs are more general than foreign keys, we here distinguish these constraints even if they are used equivalently.

- **Degree of normalization of the resulting schema:** The degree of normalization achieved, indicated in Column 5, depends on the set of dependencies taken into account. If only those FDs are considered which are directly implied by the cardinality constraints and the key definitions, then it follows from [73] that an EER-schema satisfying some preconditions, e.g. unique naming of all attributes as well as entity- and relationship types, can be represented by a BCNF relational schema. If also FDs between non-key attributes are taken into account, normalization can be done prior to a transformation at the ER level [28, 66], or after a transformation by normalizing the resulting relation schemas using well-known algorithms, e.g. Bernstein's synthesis procedure [6]. We here consider only the degree of normalization of the relational schema directly after the transformation as mentioned in the underlying approach.

- **Information preservation of the transformation:** Column 6 states whether all constructs and constraints of a given ER schema are considered during the mapping.

- **Canonical transformation:** A transformation is canonical, if for every type of the source schema a separate relation is defined. Such a transformation has the advantage that the information preserving property can easier be satisfied than using other options, and that the transformation result is more transparent and easier to understand for a user; a disadvantage is the resulting redundance. Markowitz et al. [73] combine the advantages of canonical and
non-canonical transformations, since they first transform canonically and then merge the resulting relations w.r.t. structural properties.

The other properties from Section 3 which are not mentioned here explicitly are the same for all approaches; more precisely, no approach requires user interaction, the transformation methodology is the same for all approaches, and operations are only considered in the approach of Bertaina et al. [7].

We now comment further on the approaches summarized in Table 2. Batini et al. [5] use an enhanced ER model with complex attributes and generalization structures for conceptual design. Before a transformation can be performed, all enhanced structures, e.g. generalization hierarchies and complex attributes, are simplified to entity- and relationship-type structures. Several possibilities for the simplification of generalization structures are proposed. After simplification the general transformation approach is applied, as described above for Min-Max-cardinalities.

The approach of Bertaina et al. [7] is the only one incorporating processing requirements into a transformation. To this end, a given ER schema is first simplified s.t. it consists of strong entity types and binary relationship types only. Then the schema is refined and restructured w.r.t. the processing requirements, e.g. attributes are duplicated and entity-types are split, and afterwards mapped into a relational schema using the general transformation rules for 1:N-cardinality constraints. The resulting schema is generally not normalized, but optimal with respect to the specified operations. Dogac et al. [33] derive insert and delete triggers during the translation. These triggers can be considered as a substitute for inclusion dependencies or foreign keys, which are implied by the structure and cardinality constraints of the given ER schema. The approach is vastly heuristic in nature; in particular, trigger derivation is not made formally precise.

The benefit of the approach of Elmasri and Navathe [36] is the mapping of enhanced EER structures, like the category construct or shared subclasses. The only abstraction which they do not use in their source model is the aggregation or relationships over relationships, resp. Ling [66] introduces a normal form for ER schemas. The source schema is required to obey this normal form; hence, normalization preprocessing must be performed. The transformation into the relational model results in relational schemas in 3NF or in 5NF. In contrast to most other approaches, the ER model used here supports the specification of non-key based functional dependencies and several keys for each entity type, which are considered in the transformation. Mannila et al. [68] define an Entity-Relationship Normal Form (ERNF) for relational schemas, and prove, for example, that their transformation result is in BCNF. The mapping they use is based on that of Markowitz et al. [72]; however, only FDs that are directly derived from cardinality constraints and keys are considered.

The latter is also true for the approach of Markowitz et al. [73]. They realize a transformation in four steps: First, the given ER schema is mapped to a canonical relational schema with FDs, INDs, and existence constraints. In this schema each entity or relationship type is represented by an individual relation. Second, a name assignment algorithm for the attributes is applied, to avoid ambiguities and to satisfy various assumptions, including the universal schema assumption, the universal role assumption and the one flavor assumption. Third, the schema is normalized into BCNF, and finally schemas are merged to remove redundancies. The resulting schema is still in BCNF. The approach is mathematically founded,
and the correctness of each step can be proved. Definitions are given for “equivalent information capacity” of relational schemas and for a correct transformation. The approaches described in [69, 72, 70] can be seen as predecessors of this approach; an inverse transformation for the approach is given in [69].

Springsteel and Chuang [95] also investigate the normalization of resulting relational schemas depending on the structure of the ER source schema; the proposed transformation is based on [70]. In the approach of Storey [97] various possibilities to translate generalization structures are presented, as described in the general transformation approach above. The transformation is embedded into a methodology for database design which starts with the identification of basic ER constructs and ends with a normalization of the result. Teorey, Yang, and Fry [101] derive keys, FDs, null-value constraints and foreign-key constraints from the initial ER schema. Relational normalization is considered an extra step after the transformation. The transformation itself is not information preserving, since exclusion dependencies implied by the generalization structures are missing. It should be mentioned that this approach as well as the ones of Storey [97] and Markowitz and Shoshani [73] appear to be best suited for practical usage.

4.2. From ER to OO

As was mentioned earlier, there is an increasing interest today in using OO database technology, which triggers a need for suitable techniques for OO database design. In this subsection, we survey relevant approaches which try to re-use past experience by using an ER model for conceptual modeling, and by transforming a resulting schema into the structural part of an OO schema declaration. Such transformation approaches have been reported in [8, 35, 36, 43, 60, 76, 77, 98]; approaches using another semantic data model for OO database design can be found in [11, 47, 84]. A justification for using the ER model in this setting is that it still appears appropriate, at least as long as OO models on which current systems are based have limitations w.r.t. the modeling of structural and behavioral aspects of database applications [98]; for example, they are often unable to model n-ary relationships or integrity constraints, e.g. existence constraints or keys, explicitly.

A major novel aspect in transforming an ER schema into an OO model, which has no counterpart in relational transformations, is that it now becomes necessary to define (and integrate) operations or methods which represent integrity constraints that cannot be represented explicitly via structures. For example, most OO models do not have a possibility to declare keys or more general FDs. However, such constraints can now be coded into methods, which when integrated appropriately into type or class declarations can be maintained by the underlying system automatically.

As for the relational case, a basic methodology of how an ER-schema can be mapped into an OO schema can be exhibited. It considers only the definition of the type structure without operations (since for the latter there is no commonly accepted approach yet). This methodology, which we again call ‘general transformation’, is vastly supported by most of the approaches surveyed. The integration of integrity constraints which cannot be represented explicitly through the type structure will be described in the context of specific approaches, since there is no commonly accepted methodology for this either.
4.2.1 The general transformation approach

The transformation of basic ER structures is similar to the various proposals that have recently been made. For a class-based target model, e.g. $O_2$ [3] or the C++ interface of ODMG-93 [21], structures of the ER-model can be transformed in the following way:

- For each entity-type an class is created. The attributes of the entity-type become the attributes of that class, where complex attributes can be represented directly using the available tuple, set, bag, or list constructors.

- **Binary relationship-types** can be represented by object-valued attributes in the classes corresponding to the components of the relationship-type, or by an individual class. In the first case the attributes are defined as set- or single-valued depending on the cardinality constraints defined for the relationship type. This option is particularly meaningful if the relationship type in question has no local attributes. The attributes representing a relationship type should, if possible, be defined as inverse to each other in the two relevant classes, in order to enforce referential integrity. If a path from one class to the other is relevant for the underlying application only, it will be sufficient to include such an object-valued attribute in one component class (not two), which then references the other.

- In the second case two single- and object-valued attributes are defined for the class representing the relationship type, which reference the classes corresponding to the relationship components. In the latter, object-valued attributes are included referencing the relationship class. Each pair of attributes representing such a connection should again be defined as inverse of each other; the cardinality of the attributes in the component classes is again dependent on the cardinality constraints defined for the relationship type, as in the first case. This transformation option is meaningful if the relationship type has attributes.

- **Weak entity types** which are not components of another relationship type can be represented as (possibly tuple- and set-valued) attributes in the class representing the identifying type. Alternatively, they can be represented as individual classes, and the existence relationship is manifested via (inverse) object-valued attributes as described above.

- **Recursive relationship types** are represented as object-valued attributes referencing the class in which they are themselves included.

- All other relationship-types and aggregations are mapped into a separate class, with appropriate object-valued attributes referencing the components of the relationship in question. The proceeding is similar to that described for binary relationship types, which are mapped into an individual class.

- **IS-A-relationships and generalization structures** are modeled via inheritance links between the classes representing the supertype and the subtype(s) in the ER-model. Additional ‘intersection classes’ have to be defined between two subclasses if the OO model requires every object to be a member in exactly one class and the subclasses of a generalization structure are overlapping.

- **All other constraints and structures** of the ER-schema, which cannot be described implicitly or explicitly in the OO-schema, are represented as operations or methods attached to their corresponding class. If the target OO model supports integrity constraints, e.g. keys and inverse attributes in ODMG-93 [21], they can be used instead of attaching integrity methods.

All of the above mentioned approaches generally follow this transformation methodology;
they vary a lot, however, in the way they take integrity constraints into account, if they do this at all. Other differences concern the representation of binary relationship types and the specific target model; specifics appear in the next subsection.

4.2.2 Details

In Table 3, we survey several approaches to OO transformations which exhibit a variety of target models. In this table, we again show the ER model used, where the classification follows that of Table 2 (with the exception of ‘7’ which now stands for ‘derived attributes’). Some of the approaches use a virtual OO model, in the sense that they only rely on features occurring in most OO models.

We mention that none of these approaches requires special prerequisites or user interaction. Only Nachouki et al. [76] consider processing requirements during their transformation. Except for Biskup et al. [8], nothing can be said about the correctness of the approaches, since the descriptions are vastly informal. The quality of an OO schema is discussed in Nachouki et al. [76], Kilian [60] and Biskup et al. [8]. The first concentrates on class identification, the second on the correct representation of generalization structures, and Biskup et al. investigate how an OO schema which is a result of an initial transformation can be improved by merging and decomposing classes.

We now comment on the various approaches in further detail. Biskup et al. [8] use F-logic [59], a logic-based formalism for specifying and reasoning about OO concepts, supplemented by a notion of semantic constraints for their transformation. In particular, a given ER schema is first transformed into an abstract OO schema in F-logic, which is then improved by decomposing and merging classes and determining minimal keys. The resulting schema is then mapped into a concrete OO-schema, in this case using the ONTOS system [83], which can finally be optimized w.r.t. the specific system properties at hand.

Elmasri and Navathe [36] propose a general transformation methodology, using an EER schema as source and a virtual OO model as target. They only focus on the mapping of basic ER structures, similar to the general transformation approach above. The transformation of integrity constraints outside the type structure is not considered. Elmasri et al. [35] use an enhanced EER model [36] as source and again a virtual OO model as target. Besides the mapping of EER structures, this approach focuses on the automatic generation and integra-

<table>
<thead>
<tr>
<th>Approach</th>
<th>ER source model</th>
<th>OO target model</th>
</tr>
</thead>
<tbody>
<tr>
<td>Biskup et al. [8]</td>
<td>1,3,4</td>
<td>F-logic/ONTOS</td>
</tr>
<tr>
<td>Elmasri et al. [35]</td>
<td>1,2,4,6</td>
<td>none specific</td>
</tr>
<tr>
<td>Elmasri/Navathe [36]</td>
<td>1,4,6</td>
<td>none specific</td>
</tr>
<tr>
<td>Gogolla et al. [43]</td>
<td>1,3,4,6,7</td>
<td>TROLL light</td>
</tr>
<tr>
<td>Kilian [60]</td>
<td>1,2,4,7</td>
<td>none specific</td>
</tr>
<tr>
<td>Nachouki et al. [76]</td>
<td>4</td>
<td>O₂</td>
</tr>
<tr>
<td>Narasimhan et al. [77]</td>
<td>4,6,7</td>
<td>none specific</td>
</tr>
<tr>
<td>Tari [98]</td>
<td>1,5,6</td>
<td>O₂</td>
</tr>
</tbody>
</table>
tion of methods which enforce integrity constraints given implicitly and explicitly in the EER schema, in particular keys, cardinality constraints, and existence constraints derived from generalization structures. To this end, the basic methods Constructor, Destructor, Updater, Relater, and Unrelater are defined in each class, which integrate all specified integrity constraints. The main focus lies on a transformation of different kinds of generalization/specialization constructs.

Gogolla et al. [43] use TROLL light [42] as target model. Beside other features, this model provides possibilities to represent static integrity constraints, derived attributes, and behavior explicitly. Objects are described in Troll light using templates. Constraints can be specified in a special template field using an SQL-like query calculus. The focus of the approach is on the transformation of basic constructs, and on the mapping of constraints is just mentioned. Kilian [60] investigates the differences between OO database design and ER design, and derives an OO representation of ER structures from that. The focus is on the representation of categorization constructs in an OO schema; he distinguishes subset and subtype relationships, and categorization in the sense of [37]. For the latter, he introduces an OO analogue to the category type.

Besides structural transformation, Narasimhan et al. [77] focus on the integration of constraints defined for an ER schema into an OO schema. They suggest to create a special constraint class with a subclass for every class in the OO schema, where the latter comprises all constraints of that class. Thus, two class hierarchies are obtained, one for structure, the other for constraint methods; both are in a one-to-one correspondence. The structure mapping is similar to the general approach above, but binary relationships are always represented as object-valued attributes.

The approach of Nachouki et al. [76] is unique in that it considers the design of logical access paths: First, an initial OO skeleton schema is created from a given ER diagram. Then, logical access paths of operations in the initial schema are determined, and for each a query tree is built. The latter are then merged, which results in a schema with access paths. In a further step, this schema is made acyclic. Finally, classes of the schema are merged w.r.t. the access paths, and the resulting classes can be coded in the language of $O_2$ [3]. Tari [98] uses the ERC + model [94] for structural modeling, together with an extended first-order logic for behavior modeling. The target OO model is again $O_2$ [3]. Besides a canonical structural mapping, the approach also includes a mapping of behavior into $O_2$C code.

In summary, it can be said that there seems to be a vast agreement on how to transform structural aspects from ER to OO; minor differences remain w.r.t. the way in which binary relationship types are represented, and to what extent generalization structures are considered. A major difference between the various approaches is in the handling of integrity constraints, which is generally poor. It should also be noted that only the approaches of Elmasri [35,36], Narasimhan et al. [77] and Biskup et al. [8] present methodologies which are independent of specific OO models.

Finally, there exist several approaches based on a semantic model vastly similar to the ER-model; these can be found in [11, 47, 84]. The approach of Bouzeghoub and Metais [11] is of particular interest, since here a semantic model, augmented with rules specified in first-order logic, is mapped into an $O_2$ schema and into $O_2$C methods. Additionally, a proposal for the integration of methods is made, namely to define a General Constraint.
Method for each class, responsible for invoking all operations representing integrity constraints. Thus, each update operation needs to know this general method only, and integrity operations can easily added to or deleted from a class description.

4.3. From ER to network

Transformation approaches for transforming an ER schema into the network model can be found in textbooks [5, 36, 61, 103, 105] as well as in original publications [7, 13, 22, 30, 33, 34, 53, 107]. A difficulty of this transformation direction is that the network model provides only limited possibilities to define explicit integrity constraints; almost all such constraints must be realized through application programs. Nevertheless, the network model is similar to the ER model in expressive power; therefore, a mapping can be done straightforwardly. Most approaches in this context follow Chen [22]; they differ mainly in the way they treat recursive relationships. Thus, we can again exhibit a general methodology, on which most approaches follow. We use the 1:N-notation to describe cardinality constraints.

4.3.1 The general transformation approach

Basic ER structures can be mapped as follows:

• For each entity type a record type is created. All attributes of the entity type are included as data items in the record. The keys of the entity are each defined as a unique data item group in the record.

• For each weak entity type W with identifying entity type E, a record type Rec_W and a set type is created, with Rec_W as member and Rec_E as owner. The key field of the owner record type representing E can be duplicated in Rec_W, and a Check or Structural constraint can be defined to make the foreign-key relationship explicit.

• For each binary 1:N-relationship type or binary 1:1-relationship type R with participating entity types E_1 and E_2 and corresponding record types Rec_{E_1} and Rec_{E_2}, a set type with owner Rec_{E_1} and member Rec_{E_2} is created. If the relationship type is of the form 1:1, that record type Rec_{E_i}, i \in \{1,2\}, should be chosen as member whose corresponding entity-type R_i has a mandatory participation in R; if both components have optional participation, the choice is arbitrarily. If both components have a mandatory participation and do not participate in other relationships, R can also be represented by merging Rec_{E_1} and Rec_{E_2}. For a 1:N-relationship type, the record type corresponding to the component at the 1-side becomes the owner, the other the member of the set.

A mandatory participation of entity type E in relationship type R can be represented through a mandatory or fixed set retention option of set S; a mandatory participation of the corresponding owner can only be realized through application programs.

Relationship attributes of R are included as data items in the member record type. Key fields of the owner can be duplicated as described above.

• For each aggregation, binary M:N-, or k-ary relationship type R, a link record type Rec_R is defined. Rec_R is connected to the record types corresponding to the components of R as shown in Fig. 5 for an N:M-relationship type. The set retention options must be defined as
mandatory or fixed, since $R$ is existence-dependent on its components. Key fields of the owners can be duplicated as described above.

- **Recursive relationship types** can be handled in various ways, as shown in Fig. 6. The references given below the record structures indicate where the different approaches are proposed. The example describes a relationship in which each employee is managed by one other employee, but can himself manage several other employees.

- **IS-A relationship types** are mapped in the same way as 1:1-relationship types. The options described earlier for the relational transformation, in particular those concerning the merging of generalization hierarchies, can also be applied here in a similar fashion.

### 4.3.2 Other approaches

The approaches of Bertaina et al. [7] and of Irani et al. [53] extend the general transformation approach described above. Both consider the design of an ‘optimal’ schema w.r.t. processing requirements. To this end, the former realizes a 3-step transformation: First, an initial EER schema is simplified by replacing structures which cannot be transformed directly by binary relationship and entity types. This includes the substitution of generalization and subset hierarchies as well as the conversion of recursive and $k$-ary relationships (where $k \geq 3$). Second, the simplified ER schema is improved w.r.t. performance, by partitioning entity types and replicating attributes. The resulting schema is then translated directly into a network schema via the general approach.

![Fig. 5. Representation of an $N:M$-relationship-type in the network model.](image)

![Fig. 6. Possibilities to represent recursive relationship types in the network model.](image)
4.4. From ER to hierarchical

Approaches for the transformation of an ER schema into a hierarchical one can be found in textbooks [5, 36, 61, 103, 105] as well as in original publications [23, 33, 34, 52, 81, 90, 91]. Here, mapping is not as straightforward as for the network model, since hierarchical structures make it more difficult to map the ER structures of higher abstraction. In addition, the design of a hierarchical schema is strongly dependent on the specific target DBMS and on the operations that should be performed on the database, since logical and physical schemas are not separated. Due to these reasons, most transformation approaches are basically design methodologies which are independent of a concrete system. Only the transformations described in [52, 81, 90, 91] take processing requirements into account and consider the design of an optimal hierarchical schema. As before, we first describe a general transformation methodology and then discuss approaches of additional interest.

4.4.1 The general transformation approach

Basically, the structures of an ER schema are mapped in the following way:

- **Entity types, binary 1:1- and 1:N- as well as IS-A relationship types** are mapped as in the network model, with the difference that parent-child relationships (PCRs) are not named and that the relationship component which corresponds to the child in a PCR has mandatory participation. If the latter should not hold, these relationship types must be treated like an N:M-relationship type as described next.

- **N:M-relationship types** can be transformed in various ways, depending on whether virtual PCRs are allowed. For the case that they are, the examples from Fig. 5 are restated in Fig. 7, where rectangles represent record types, arrows denote virtual PCRs, and lines indicate physical PCRs. Examples for the case that virtual PCRs are forbidden are shown in Fig. 8. In this case, N:M-relationships are realized through a duplication of member record occurrences or through the creation of a new root record type — called ‘intersection record’ — which connects the records corresponding to the relationship components.
The transformation of \textit{k-ary relationship types} and of \textit{aggregations} is similar to that of \textit{N:M-relationship types}. Options are illustrated in Fig. 9, which describes a ternary relationship type \textit{shipment} between entity types \textit{Part}, \textit{Project}, and \textit{Supplier}.

- \textbf{Recursive relationships} cannot be represented directly in hierarchical structures. Therefore, a widely used method is to substitute a given recursive relationship type, by copying the entity type in question and giving the copy a role name. The new entity type is then connected through a relationship type representing the recursive relation with the original entity type. The transformation of the restructured schema can then proceed as described above. Note that recursive relationships are often ignored; on the other hand, the methodology just sketched is independent of the target data model.

When converting each relationship type and entity type as described above, a forest consisting of physical and virtual PCRs and of record types is obtained. These trees can eventually be combined based on common nodes. If the resulting schema should consist of only one tree, a new root needs to be created which connects to the roots of all 'local' trees. Basically all of [23, 33, 34, 36, 61, 103, 105] support some version of this general approach. Only [34] describes an algorithm explicitly; [33] shows how generalization hierarchies can be represented in the hierarchical model. Another possibility to produce a hierarchical schema appears in [5]; the idea is to transform into a network-like schema first and from there to the hierarchical model.

\section*{4.4.2 Other approaches}

Only a few approaches go beyond the general approach just described, including [52, 81, 90, 91]; they consider processing requirements. The general idea is to first determine for each relationship type all possible transformations, called \textit{local} transformations. Next, the local transformation optimal w.r.t. given processing requirements is determined for each relationship type; finally, the optimal local transformations are merged into the result. The approaches mentioned vastly follow this methodology, and differ mainly in the integrity constraints considered during the transformation.

In Sakai's method [90, 91], the integrity constraints which are considered within local transformations are FDs, existence dependencies, and first-order hierarchical dependencies (see Subsection 6.6). If there is a relationship \textit{R} with associated entity types holding none of these dependencies, local translation is done w.r.t. 'access weights'. Such a weight is calculated for every pair of an entity and a relationship type in the schema, by considering how, how often and how important an access to an entity is through a relationship. This is done in a similar way in [52]. The transformation of Navathe and Cheng [81] is based on the
approach of Sakai [90]; it uses an extended ER model including subtype-supertype relationships, generalization hierarchies, and aggregation. It differs from the above in that various possibilities for the translation of aggregations and generalizations are considered. Integrity constraints taken into account for local translation are FDs, transitive dependencies, and first-order hierarchical dependencies.

5. Reverse engineering: From a logical model to ER

As was mentioned in Section 2, reverse engineering is of increasing importance these days. However, reverse engineering transformations [2, 27, 44, 45, 46] are not as straightforward as forward engineering ones. Indeed, each source schema can typically be transformed into several different resulting schemas, and the major difficulty is to choose the schema that gives the best conceptual view on the modeled application. User interaction is commonly necessary to achieve this task. We now survey various approaches for reverse engineering, where we follow the same order of logical data models as in the previous section.

5.1. From relational to ER

Algorithms for transforming a relational schema back into the ER model can be found in some textbooks [5, 68] as well as in a number of recent papers [18, 24, 25, 26, 29, 34, 41, 56–58, 70, 69, 80, 96]. Approaches using the Binary Relationship model, which is similar to the ER model, appear in [17, 93]; we do not discuss them further in this paper.

Up to now, a generally accepted methodology for reverse engineering has not been established, but the approaches we found can roughly be classified into two categories as follows:

Approaches in the first class elicit the semantics of the given relational schema through an evaluation of its inclusion dependencies (INDs). Each IND is interpreted with respect to the role (key, part of a key, foreign key, non-key) of its attributes. Most of these approaches require a complete set of INDs and keys of the relations; some take only key-based INDs into account [18, 56, 69], while others consider general INDs [57, 68]. As a result, some approaches are inapplicable without explicitly given sets of INDs and keys.

Approaches in the second class derive an ER schema through an evaluation of keys, their construction through other keys, and a discovery of foreign keys. The name semantics plays an important role in such approaches, since relationships between keys are mainly identified through their names. Therefore, a proper name assignment, especially for all key attributes, is essential. Several of these approaches also classify the relations of the source schema with respect to the construction of their keys, e.g. whether the primary key of a relation is the concatenation of the primary keys of other relations. The transformation is then done on the basis of this classification. The approaches are applicable provided only keys and attribute names are given. The evaluation of the name semantics can be considered as a heuristic to derive foreign-key constraints and hence key-based inclusion (since the former are a special case of the latter). Thus, approaches in this second class are more heuristic in nature than
approaches of the first; it includes [5, 29, 34, 41, 58, 70, 80]. Finally, the approaches of Chiang et al. described in [24–26] fall into both classes.

5.1.1 Survey
The approaches to reverse engineering mentioned above are surveyed in Table 4 with respect to the following issues:

- **Target ER model**: The more modeling structures the target model contains, the more complicated does reverse engineering get. Column 2 lists extensions of the ER model that are used besides standard ER structures.
- **Principle transformation method**: Column 3 lists the principle design method used according to the above mentioned classification.
- **Prerequisites**: Column 4 lists prerequisites required for a transformation. These include proper naming of attributes, availability of all keys and INDs, a set of FDs, a database instance, and the required normal form of the relational schema.
- **Transformation properties**: The properties considered in Column 5 include derivation of cardinality constraints, consideration of all given keys, identification of non-standard relationships, e.g. optimization structures, consideration of the synonym/homonym problem, the existence of a pseudo-code algorithm and a complexity evaluation of the algorithm.
- **User interaction**: Some approaches require user interaction (indicated in Column 6 by ‘Y’), while some do not.

All approaches based on INDs do not consider the problem of homonyms and synonyms.

<table>
<thead>
<tr>
<th>Approach</th>
<th>ER-model</th>
<th>Method</th>
<th>Prerequisites</th>
<th>Properties</th>
<th>User</th>
</tr>
</thead>
<tbody>
<tr>
<td>Batini [5]</td>
<td>G, S</td>
<td>KEY</td>
<td>Keys, 2NF/3NF</td>
<td>Cc, Mk, Hom</td>
<td>Y</td>
</tr>
<tr>
<td>Casanova [18]</td>
<td>INC</td>
<td></td>
<td>Keys, KID</td>
<td>Pk, Ps</td>
<td>N</td>
</tr>
<tr>
<td>Dumpala [34]</td>
<td>KEY</td>
<td></td>
<td>Keys, FD, 3NF, NA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Davis [29]</td>
<td>KEY</td>
<td></td>
<td>Keys, 3NF, NA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Fonkam [41]</td>
<td>S</td>
<td>KEY</td>
<td>Keys, 3NF</td>
<td>Cc, Mk, Hom</td>
<td>Y</td>
</tr>
<tr>
<td>Ji [56]</td>
<td>A, S</td>
<td>INC</td>
<td>Keys, ID, 3NF</td>
<td>Cc, Mk, Opt</td>
<td>Y</td>
</tr>
<tr>
<td>Johanneson [57]</td>
<td>S</td>
<td>INC</td>
<td>Keys, KID</td>
<td>Mk, Ps, Com</td>
<td>N</td>
</tr>
<tr>
<td>Kalman [58]</td>
<td>S</td>
<td>KEY</td>
<td>Keys, 3NF, NA</td>
<td>Cc, Mk, Opt</td>
<td>Y</td>
</tr>
<tr>
<td>Makowsky [70]</td>
<td>S</td>
<td>INC/KEY</td>
<td>Keys, KID, NA</td>
<td>Cc, Pk, Ps</td>
<td>N</td>
</tr>
<tr>
<td>Mannila [68]</td>
<td>S</td>
<td>INC</td>
<td>Keys, FD, ID</td>
<td>Cc, Mk, Opt, Ps</td>
<td>Y</td>
</tr>
<tr>
<td>Markowitz [69]</td>
<td>A, S</td>
<td>INC</td>
<td>Keys, FD, KID</td>
<td>Cc, Pk</td>
<td>N</td>
</tr>
<tr>
<td>Navathe [80]</td>
<td>G, S</td>
<td>KEY</td>
<td>Keys, 3NF</td>
<td>Cc, Mk, Hom</td>
<td>Y</td>
</tr>
</tbody>
</table>

Model: S = Subtype/Supertype relationships G = More complex generalization structures A = Aggregation or relationships over and relationships

Prerequisites: NA = Proper name assignment Keys = Keys ID = Set of INDs KID = Set of key-based INDs FD = Set of functional dependencies NF = Assumed normal form Inst = Database instance

Method: KEY = Key-based approach INC = IND-based approach

Properties: Cc = Derivation of cardinality constraints Pk = Only consideration of primary keys Mk = Consideration of candidate keys Hom = Problem of homonyms/synonyms is considered Opt = Identif. of 'non-standard' relationships Ps = Pseudo-code algorithm Com = Complexity evaluation is given
explicitly, since they assume that all relationships between relations are given via the INDs; 
therefore, no relationships have to be derived by evaluating the attribute name semantics, as it 
is done by the approaches following the key methodology. They consider the problem implicitly, 
since the attributes on the left and right side of an IND are assumed to have the 
same meaning (and are therefore synonyms in case of different naming). Therefore this 
property is not included in the table for these approaches.

5.1.2 Comments

The approaches of Batini et al. [5], Navathe [80], and Kalman [58] are all based on the one 
by Dumpala [34]; in all of them, relations are classified w.r.t. their primary keys. Such a 
classification is obtained via an interaction with the user. The relational schemas are then 
interpreted and transformed based on the classification. Cardinality constraints are derived via 
an evaluation of FDs. The relations are assumed to be in 3NF, except for the approach of 
Batini [5], who allows relations to be in 2NF only. All these approaches, except [34], consider 
candidate keys in a preprocessing step by replacing the primary key by a candidate one in 
special cases. Kalman [58] even describes an implementation.

Other approaches based on the key method, but without relation classification, are those by 
Davis and Arora [29], and by Fonkam and Gray [41]. The former are the only ones to propose 
an algorithm for deriving behavior, in the form of insert and update operations, during the 
translation. The benefit of the approach of Fonkam and Gray [41] is that all candidate keys 
are fully considered during for the transformation, which is not the case for the other 
key-based approaches; additionally, a comparison is made between the approaches of Davis 
and Arora [29], Johanneson and Kalman [57], and Navathe and Awong [80].

The approach of Chiang et al. [25] requires the existence of a database instance as the only 
prerequisite. It is a mixture of the above described key-based and IND-based approaches in 
that first the relations are classified similar to [80], and then INDs are generated by applying 
heuristics. The latter are verified w.r.t. the given database instance, and finally an ER schema 
is derived from them.

The approaches of Casanova et al. [18], Ji [56], Johanneson et al. [57], Markowitz et al. 
[69], and Mannila et al. [68] use INDs to extract an ER schema from a relational schema, 
where some allow general INDs, while others consider key-based ones only (see 5.1). All 
approaches require a set of INDs as initially given. [18] and [68] introduce an Entity– 
Relationship Normal Form (ERNF) for transformation purposes, but only the latter reference 
does preprocessing to convert a given relational schema into ERNF; consequently, their actual 
transformation requires an ERNF relational schema.

A similar approach, using an ER normal form, is given by Makowsky et al. [70]. Their 
transformation is again a mixture of a key-based and an IND-based approach. First, an initial 
ER schema is produced w.r.t. the given keys; Then, the resulting schema is then verified 
based on the given INDs. We finally mention that another possibility for transforming a 
relational schema into an (extended) ER schema is to first extract all FDs from the relational 
schema and then to derive an ER schema with respect to a minimal cover of these 
dependencies. However, this requires that the relational schema satisfies the universal relation 
schema assumption and contains no synonyms; if this is the case, the methodologies of Briand 
et al. [14] or Jajodia [54] can be applied.
5.2. From O0 to ER

Apparently no proposal has yet been made for transforming an OO schema into the ER model. A reason for this may be that it is not considered relevant; indeed, since OO database systems are newcomers in the field, a need for doing reverse engineering on them does not yet exist. However, as the number of installations of such systems will grow over the next years, even this transformation direction will become increasingly important. It is worth noting that for OO database systems supporting ODMG-93 [21], such a transformation can easily be specified in a way similar to relational reverse engineering. The major problem is to discover those classes which represent k-ary relationship types instead of entity-types; this can be done by an evaluation of the keys of each class. In addition, each (pair of) object-valued (inverse) attribute(s) can be represented as a relationship type, every class as an entity type, and inheritance structures are mapped to generalization structures. Classes representing a relationships are transformed to relationship types, and the object-valued attributes in these classes to links in the ER-schema. This basic transformation methodology can easily be extended.

5.3. From network to ER

Approaches for the reverse engineering of network schemas back into ER schemas can be found in the textbook [5] as well as in [10, 34, 75, 107]. The major difficulty that arises here is to decide which record type represents an entity and which one represents a relationship or a link record, respectively. If network schemas are ‘well-formed’ [10] in the sense that
• each record type has an identifier specified,
• each record type contains only data items describing properties of the objects represented by the record type,
• each set type has an insertion clause with structural insertion mode,
• all existence dependencies are specified by set-retention options,
then the transformation is straightforward, since the necessary distinctions can easily be made. However, network schemas are rarely well-formed; instead, the relevant information is often buried in application programs. User interaction or the use of heuristics is then necessary to elicit the semantics of a given schema.

The characteristics of the different approaches mentioned above are as follows: Batini et al. [5] use an extended ER model including generalization and specialization structures as target model. The transformation of a network schema is done in four steps: First, every record type is mapped into an entity type. The fields of the record type become the attributes of the entity type. Second, all those entity types are identified which actually represent relationship types. Third, every set type is mapped to a 1:N-relationship type, except for those which point to a record type whose corresponding entity type was previously identified as a relationship type. Fourth, generalization structures and weak entities are identified. However, it remains open how to distinguish relationship types from entity types and how to identify generalization structures as well as weak entities; therefore, this approach is hardly practical.

Boulanger and March [10] realize a transformation in two steps. First, they extract all FDs as well as multivalued dependencies from the given network schema, using key-field
declarations in the record and set types. These dependencies are verified by the user, and a
minimal cover is calculated to remove redundant dependencies. Third, an ER schema is
derived from the dependencies, e.g. for every FD with only one attribute on the left-hand side
an entity type is defined, and for each other dependency a relationship is created. It turns out
that the same sequence of steps can also be used in the context of the relational model.

Dumpala and Arora [34] use a network model supporting N:M- and recursive set types. Their transformation is simple and straightforward: Each record type is converted into an
entity type, each set type into a relationship type with the same cardinality constraint, and
each recursive link into a recursive relationship type. However, this transformation is not
appropriate for network models supporting 1:N-set types only. Michaely and Scheer [75]
realize the transformation through an evaluation of set-removal constraints together with the
number of set types pointing to a record type. Their approach can even do without user
interaction in special cases; however, ambiguities cannot be resolved algorithmically (e.g.
when a record type has more than one owner and the corresponding set removal constraint is
fixed or mandatory). Wong and Katz [107] require a classification of the given record types as
starting point; however, since that can only be derived from a previous ER to network
transformation, their approach is not suitable for a reverse engineering of network schemas.

With respect to the transformation properties listed in Section 3, the following can be stated
for these approaches: Only Batini et al. [5] use an extended ER model supporting
generalization, while all others support the classical ER model only. All approaches require
given record and set types, but only Boulanger et al. [10] and Michaely and Scheer [75] use
given set type options to derive semantically meaningful conclusions from them. Cardinality
constraints in form of the 1:N-notation are derived by each approach, and the user is involved
only in the approaches of [5, 10, 75]. The approach of Dumpala and Arora [34] can yield
schemas of bad quality since even link-record types representing a relationship type are
mapped to an entity type.

5.4. From hierarchical to ER

Approaches to transform a hierarchical schema into the ER model can be found in [5] as
well as in [16, 34, 80, 106]. One difficulty of this transformation direction is that most integrity
constraints and structural information are hidden in application programs which can only be
evaluated by a user. The characteristics of these approaches are as follows:

Dumpala and Arora [34] provide a simple two-step procedure which is independent of a
specific hierarchical system: First, all trees in the hierarchical schema are connected with
respect to common, but eventually renamed record types. The result is a network-like schema.
Second, all record types are replaced by entity types, and all parent-child relationship types
are replaced by relationship types with cardinality constraint 1:1 or 1:N. Clearly, this
algorithm can produce redundant ER schemas of low quality.

A better approach is proposed by Navathe and Awong [80]; it takes cardinality constraints
into account and produces 'well-designed' ER schemas with respect to the properties
described in Section 3. The transformation is done in two steps: First, a preprocessing step
produces a network-like structure. In this step, duplicate records types as well as all virtual
pointers are eliminated from the schema, and the remaining record types are classified into
one of the following types: \( M:N \)-records, entity records, subclass records, relationship records. Additionally, cardinality information is extracted for each link between two record types. Most of this information must be provided by the user. In the second step, the record types are converted into structures of the ECR model \[36\]. This is done with respect to the classification and the cardinality constraints of the links; for example, each linked pair of \( M:N \)-records is mapped into two entity types with an \( M:N \)-relationship type between them, or each subclass record is mapped to a subclass category.

Winans and Davis \[106\] propose an approach to map an IMS schema into an ER schema without user interaction. They derive insert and update operations from the given schema during the mapping. Their translation procedure is quite simple: Each root record type (i.e. segment without a parent) is mapped into a self-identified entity type, whereas all other record types are transformed into existence-depended entity types. Two entity types, corresponding to a linked parent and child in the hierarchy, are connected by a \( 1:N \)-relationship type. Paired record types (in IMS terminology) are considered separately, since they are used to implement one relationship; therefore, the corresponding entity and relationship types are merged. The resulting schema consists of entity types which are connected through binary \( 1:N \)-relationship types.

Batini et al. \[5\] convert a hierarchical scheme into an extended ER schema, by first transforming it into a network schema (using the method described in Section 6.8) and then applying a transformation method for network schemas as described in Section 5.3.

With respect to the properties listed in Section 3 we can state following: Navathe and Awong \[80\] propose the only approach deriving generalization structures during the transformation. None of the approaches assumes prerequisites, and only \[80\] and \[5\] involve the user in the transformation. The former also propose the only approach yielding an ER-schema of good quality, since all others (but \[5\]) derive only binary relationship types and ignore structures specific to hierarchical systems.

6. Direct transformations between logical models

We now turn to a third way of transforming one data model into another, which differs from the previous two in that the incorporation of a conceptual model is dropped; instead, the idea now is to transform logical models directly from one to the next. Transformations of this kind seem to be of increasing practical importance, in particular in the context of system migrations or heterogeneous systems. In addition, direct transformations are theoretically appealing, since structural aspects of traditional logical models have many commonalities. We focus especially on transformations between the relational and OO models, since these are of most practical relevance. Other transformation directions are sketched for the sake of completeness.

Interestingly, only a few papers discuss the problem of schema mappings between logical models. A reason for this may be seen in the fact that practical applications prefer indirect schema transformations, such as those described earlier in this paper. On the other hand, indirect transformations are not always appropriate. For example, in a heterogeneous
environment, built upon different database systems, an indirect transformation will lose processing efficiency, since actually two transformations are made instead of one.

6.1. From OO to relational

Approaches for mapping a schema of an OO model into a relational one can be found in [51, 9]; mappings of semantic models similar to an OO model appear in [48, 67]. Transformations of this kind are quite similar to the mappings of an ER schema into a relational schema. One major difference is that objects can be identified by their unique objects IDs (OIDs), chosen independent of (attribute) values, whereas entities are commonly identified through values for their attributes.

Hull and Yoshikawa [51] propose a simple approach of how a schema of a class-based OO model supporting inheritance, aggregation, and atomic and set-valued attributes, can be represented by a relational database schema consisting of unary and binary relation schemas and a number of constraints. To this end, they define for each class in the given OO schema a unary relation schema with one attribute (called the OID attribute), which is used to represent the OIDs of that class. For each attribute A of each class a binary relation schema is introduced, which contains A and the OID attribute of this class. An FD is used to state that A is atomic (if this is the case); INDs are used to link those relation schemas which were produced for one class. Additional INDs describe inheritance structures or a foreign-key relationship resulting from the transformation of an object-valued attribute. Exclusion dependencies are used to model the disjointness of classes unrelated in the inheritance hierarchy. Clearly, the resulting schema is not a well-structured one (for example, the properties of an object typically get distributed over several relations); however, this can be overcome, for example, by a subsequent normalization.

The approach of Blaha et al. [9] describes the mapping of an OMT schema RUM91 into a relational one with key, foreign key and existence constraints. The general procedure is similar to that described in Section 4.1.1. for ER schemas. Four possible ways for the mapping of generalization structures are proposed, three of which are identical to those described in Section general. It's worth noting that here again only atomic attributes are allowed for each class.

6.2. From relational to OO

Approaches for mapping relational database schemas into OO schemas can be found in [19] [39, 85, 86]. The problems such a transformation is faced with are similar to those occurring in reverse engineering, as described in Section 5.1; indeed, they include discovering relationships (now hidden in relations), and giving a semantically correct interpretation to all relationships and structures w.r.t. the potential structures of the target model. The three approaches described in the remainder of this subsection differ in the extent to which they try to discover optimization structures, and in their degree of possible automatization. It's worth noting that all of them can easily be adapted for a reverse engineering with the ER model as target.

Castellanos et al. [19] use the BLOOM data model [20] as a target for their mapping. In a
first step, knowledge acquisition is made, where keys, FDs, INDs, and exclusion dependencies are derived from a given database instance. Additionally, the schema is normalized to 3NF, and the keys of each schema are classified into ‘proper’ and ‘extraneous’ identifiers. In a second step, the INDs are analyzed with respect to their structure, which is similar to [57] [68]; the process includes the identification of missing entities and the extraction of semantic meaning from INDs. Finally, remaining attributes not included in an IND are assigned and BLOOM classes are created. User interaction is required only for the classification of keys as well as the semantic interpretation of IND structures. The approach is mainly based on the evaluation of inclusion dependencies and their semantic meaning, but relationships which cannot be described directly by INDs are not discovered. Additionally, various optimization structures, e.g. horizontal or vertical decomposition of relations, derived or duplicated attributes, or different representation possibilities of complex attribute structures, are not considered. However, the benefit of this approach is a high level of automatization, provided a database instance is given.

Premerlani and Blaha [85, 86] put an emphasis on discovering tricky relationship representations. To this end, they suggest a seven-step procedure for transforming a relational schema into an OMT schema [89]. In the first step, an initial schema is produced by representing each relation and its attributes as a tentative OMT class. In the second and third step, groups of candidate and foreign keys are determined. Next, tentative classes are refined by merging horizontally and vertically split classes into a single class. Generalization hierarchies are identified in the fifth step using several heuristics, which are based on the representation techniques for generalization hierarchies as described in Section 4.1. In the sixth step, associations are discovered through an evaluation of keys; in the final step, the result schema is improved, e.g. redundant associations are eliminated.

In [39] we have suggested to use the ODMG-93 model [21] as transformation target. To this end, we first define a notion of structural and semantic completeness for relational database schemas. The former is based on an evaluation of the correspondence between attributes with the same meaning (including synonyms) and INDs; as soon as a given schema is structurally complete, the search for relationships between relations can be terminated. Semantic completeness of a schema requires that the (semantic) meaning of all constructs of a given schema has been clarified. This includes, for example, a classification of all relation schemas into object- and relationship-relations, and a classification of all INDs into various types; details are omitted. The transformation itself now becomes a three-step process: First, a complete relational schema is produced, i.e. the given schema is completed. This includes an identification of synonyms and homonyms, a classification of attributes into equivalence classes, the determination of keys and FDs, a particular 3NF normalization similar to [15], an analysis of INDs similar to [57, 68], a removal of redundant attributes and relational schemas, and an identification of inheritance structures. In the second step, a canonical transformation is made, which basically transforms each relation into an ODMG class. Finally, the OO schema so obtained is restructured with respect to various OO aspects, including an elimination of artificial keys, elimination of relation schemas representing binary relationships, identification of complex attribute structures, and the redefinition of objects as literals (types). The overall approach requires more user interaction than the approach of Castellanos, but it is capable of identifying more tricky and complex relationship structures.
6.3. From network to relational

Transformation algorithms for going from network schemas directly to relational ones can be found in [10, 78, 102, 107, 108]. Since both models have communalities in their structure, a transformation can be done straightforwardly. The major problem is to identify keys in the network schema. If every record type already has a key and all foreign-key relationships are specified through a 'structural' clause, the network schema can already be considered as a kind of relational schema. The characteristics of the approaches are as follows:

Navathe [78] converts a network schema into a relational schema in two steps: First, a schema diagram (SD) [79] is constructed for the underlying network, which actually is an enhanced form of a network schema containing information about the keys of the record types. More precisely, the SD states for every record type whether or not the record type needs external identification from other record types. For this purpose the set types are classified into hierarchical set types, indicating that a member record type A provides external identification of parent record type B, and non-hierarchical set types, which do not indicate external identification for the member record type. The classification of set types has to be done by the user. Second, the SD is converted into a relational schema by converting each record type into a relation. The resulting relation consists of the attributes corresponding to the data items of the record type plus the key data items from the owner records needed for the unique identification of the record. Finally, all non-hierarchical set types are transformed into a single relation containing the key attributes of the owner and member record types.

Similar to that is the method of Zaniolo [108]. In this approach, the key of every record type is determined by an evaluation of the record and the set type options. For each record type a relation is created containing the attributes corresponding to the data items of the record type and the attributes corresponding to the calculated key. All set types not necessary for the identification of a record type are realized through the insertion of a foreign key instead of an individual relation as in the previous approach. The different treatment of set types in the approaches just described are based on the fact that the former assumes every set to represent potentially an N:M-relationship because of the possible duplication of record occurrences, whereas the latter considers a set-type as 1:N-relationship. A limitation of Zaniolo’s approach is that he requires a ‘well-formed’ schema (see Section 5.3) as input. Similar are the approaches of Tsichritzis and Lochovsky [102] and of Wong and Katz [107].

The approach of Boulanger and March [10] starts by extracting functional dependencies and multivalued dependencies from the network schema and translates them into a semantic representation using the Functional Dependency Model [40]. This model has a direct translation into the relational model. The derived constraints must be verified by the user.

6.4. From hierarchical to relational

An approach for hierarchical (IMS) to relational database migration is given by Meier et al. [74]. In their approach hierarchical structures can be converted using five predefined mapping rules, which must be applied manually by a database administrator. One rule maps a single segment into a single relation. The relation gets all data fields of the corresponding segment plus the key fields of all ancestors of a hierarchical path, which includes the record type, as
attributes. Two other rules capture cases where more than one segment is represented as one relation, which can be seen as a kind of denormalization. In these cases the data and key fields of the corresponding segment become attributes of a newly created relation. The fourth rule describes a kind of normalization by representing repeating groups as individual relations, and finally, the last rule describes the splitting of one segment into two or more relations according to the values of a data field; this is again a kind of denormalization. The approach is more heuristic in nature, since it only describes the rules which can be applied, but it does not state how to put them into an algorithmic framework.

6.5. From relational to network

An approach for mapping a set of 3NF relational schemas into a DBTG network schema is given by Hevner and Yao [49]. The approach requires a knowledge of all FDs and a unique naming of attributes. The transformation is done by splitting the set of attributes of each relation into separate record types which are then linked by a set type. The splitting is done w.r.t. common attribute groups (CAGs). Each CAG describes a set of attributes which always appear together in the given relations and do appear in at least two relations. The directions of the newly generated set types are dependent on the FDs holding between two separated sets of attributes. The newly created record and set types are integrated into an initially empty network schema. When the algorithm terminates, a network structure is formed containing an individual record type for each CAG appearing in the relational schema and an individual record type for each set of attributes of any relation not contained in CAG. The resulting network schema maintains all properties of the original 3NF relational schema [31].

6.6. From relational to hierarchical

Two approaches dealing with the identification of hierarchical structures in relational databases are [32, 64]. Both approaches use as input a universal relation schema. Delobel [32] introduces the concept of first-order hierarchical dependencies (FOHDs). An FOHD $X:Y_1|Y_2|...|Y_K$ holds in a relation $R(X,Y_1,Y_2,...,Y_K)$ if

$$R[X,Y_1,Y_2,...,Y_K] = R[X,Y_1]^*R[X,Y_2]^*...^*R[X,Y_K]^*.$$ 

In other words, a $Y_i - X$ pair is independent of a $Y_j - X$ pair for $i \neq j$. It is obvious that FOHDs are strongly related to FDs and to multivalued dependencies (MVDs). Indeed, interactions between these types of dependencies as well as inference rules for FOHDs can be stated [32]. Given now a relation $R$ and a set of constraints (consisting of FOHDs, FDs, and MVDs), a set of FOHDs valid for $R$ can be derived. The relational schema is then transformed with respect to these FOHDs.

Lien [64] realizes a hierarchical decomposition of a relational schema $R$ in two steps: First, the universal schema $R$ is decomposed with respect to the MVDs holding in the schema. For this purpose, an improved version of the decomposition algorithm from [38] is used; the result is a decomposition tree. In the second step a hierarchical schema is created through an evaluation of the decomposition tree.
6.7. From network to hierarchical

A simple procedure to convert a network-like structure into a hierarchical structure is given in Batini et al. [5]. For this purpose, all record types are considered having more than one set type pointing into them; these are called 'link-records' in the following. To convert these structures into hierarchical ones, it is determined, if the hierarchy inversion heuristic applies: If a record type R (link record) has two (or more) set types pointing into it from record types S and T (and others), and S and T (and others) in turn, have no relationship pointing into them, then R can be treated as parent record type and S and T (and others) as its children; if the resulting hierarchy is meaningful and reasonable, the hierarchy inversion heuristic applies. In case the heuristic does not apply, two cases must be distinguished, depending on whether virtual relationships are allowed. If virtual PCR are allowed, the link record type is connected through a parent-child relationship (PCR) with one of its parents, and through virtual PCRs with the others. If only one virtual parent is allowed for a child record type (like in IMS), and if more than two relationships, say N, point into the link record, then N - 1 secondary copies of the link record have to be created, which are connected to the original link record through PCRs, and which are connected to the original owner record types via virtual PCRs. If virtual PCRs are not allowed, a copy of the link record type is created for each parent record type, and each copy is connected through a physical PCR with one parent record type. The result of this restructuring includes one or more hierarchies which are eventually connected through virtual relationships. If only one hierarchy is desired, a common root can be defined which is connected through PCRs to the roots of the single hierarchies.

6.8. From hierarchical to network

A simple procedure to convert a hierarchical schema into a network schema is given in Batini et al. [5]: Every parent-child relationship is converted into a named set type. Every virtual parent-child relationship is converted into a named set type with the virtual parent as an owner and the virtual child as a member. Finally, redundant records (which may not exist) are removed. However, this approach yields suboptimal results only, since special relationships, e.g. cases were a root record type states a relationship between two or more member record types (see the hierarchy inversion heuristic above), are not considered.

7. Conclusions

In this paper, we have given a detailed survey of transformation methodologies and approaches for the various directions which are relevant to conceptual and logical database design. In particular, we have surveyed forward engineering approaches which start from an ER schema and transform this into a logical data model. Next, we have discussed approaches to reverse engineering which start from a logical schema and aim at deriving an ER schema. Finally, we have considered approaches which allow direct mappings between logical models. For all three direction types, we discussed the ER, the relational, the network, the
hierarchical, as well as OO models wherever applicable. Our main goal was to identify commonalities and differences of these approaches, and to indicate the variety of methods which have been proposed to date.

It turns out that, in each case, some form of generally agreed concept can be identified, and that a number of options exist which differ in various details. The analysis we have reported in this paper in particular shows that traditional techniques for data model conversion can even be adopted to the current state-of-the-art, OO models, provided behavior modeling is not considered. It also shows the naturalness and adequateness of the concepts of ‘entity’ and ‘relationship’ as offered by the ER model; indeed, a major issue in reverse engineering is the proper identification of these concepts even in models where they are not explicitly supported.

Considering the vast amount of literature that has been published on the topic of schema transformations, it appears surprising that commercially available database design tools are still so rudimentary. Indeed, there seems to be no good reason why such tools, in particular those offered by database system vendors, still have to be so primitive. Therefore, it seems necessary to exploit the huge body of knowledge in this area for building better design tools. We finally mention that we are currently working on the development of such a tool, in which several of the approaches reported in this paper will be implemented as algorithms a user can run. The goal is to integrate this tool into a database design environment, under construction at our and other sites [4].

References


Christian Fahrner received his MSc degree (Dipl. math. oec.), from the Technical University of Karlsruhe, Germany, in 1993. He is currently working as a PhD student at the University of Münster under supervision of Gottfried Vossen, in a project on database design sponsored by the Deutsche Forschungsgemeinschaft (DFG). His research interests include data modeling and database transformations.

Gottfried Vossen received the MSc degree (Dipl.-Inform.), the PhD degree (Dr. rer. nat.), and the German Habilitation, all in Computer Science, from the Technical University of Aachen, Germany, in 1981, 1986, and 1990 resp. He is professor of computer science and a director of the Institut für Wirtschaftsinformatik at the University of Münster, Germany, after having held visiting positions at the University of California, San Diego, USA, the universities of Kiel, Düsseldorf and Koblenz in Germany from 1986 to 1991, and the position of an associate professor at the University of Giessen, Germany, from 1991 to 1993. His research interests include theoretical and practical issues as well as applications of object-oriented database systems, in particular models for data and objects, database design, languages for object-oriented databases, transaction processing in database systems, and integration of databases into scientific applications. He is an author/co-author of over 40 papers on databases and an author/editor/co-editor of 10 books on databases as well as on computer architecture.