ELSEVIER Data & Knowledge Engineering 12 (1994) 107–142

DATA & KNOWLEDGE ENGINEERING

Reverse engineering of relational databases: Extraction of an EER model from a relational database[†]

Roger H.L. Chiang[a,*], Terence M. Barron[b], Veda C. Storey[b]

[a]*Information Management Research Center, School of Accountancy and Business, Nanyang Technological University, Nanyang Avenue 2263, Singapore, Singapore*
[b]*William E. Simon Graduate School of Business Administration, University of Rochester, Rochester, NY 14627, USA*

Abstract

A methodology for extracting an extended Entity-Relationship (EER) model from a relational database is presented. Through a combination of data schema and data instance analysis, an EER model is derived which is semantically richer and more comprehensible for maintenance and design purposes than the original database. Classification schemes for relations and attributes necessary for the EER model extraction are derived and justified. These have been demonstrated to be implementable in a knowledge-based system; a working prototype system which does so is briefly discussed. In addition, cases in which human input is required are also clearly identified. This research also illustrates that the database reverse engineering process can be implemented at a high level of automation.

Key words: Database reverse engineering; Relational databases; Extended Entity-Relationship model; Database design

1. Introduction

In many organizations there are a large number of databases that have evolved over many years. A major difficulty with these databases is that often the meaning of the data has been lost so that no one knows exactly what the data and the relationships between the data really are [13]. This lack of understanding hampers the effective utilization of data within an organization [19] and also reduces the likelihood that maintenance activities can be performed correctly. Such an understanding can only be achieved by raising the level of abstraction above that of the database itself, and representing it as a conceptual model. Since Entity-Relationship models are the most widely used conceptual models, the objective of this research is to

[†] This research was supported by the William E. Simon Graduate School of Business Administration, University of Rochester. The authors wish to thank the anonymous reviewers for their insightful and helpful comments on an earlier version of this paper.

present a methodology that extracts an extended Entity-Relationship (EER) model from an existing relational database. In order to do so, concepts from software reverse engineering are applied. The methodology analyzes, not only the data schema, but also data instances which contain detailed information about the application domain.

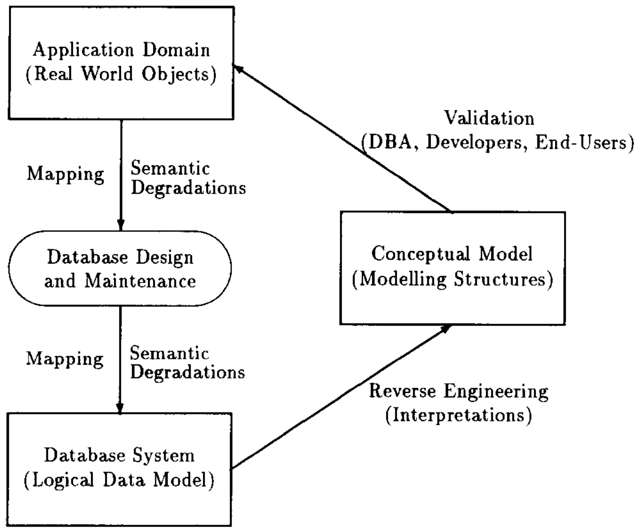
This paper is divided into six sections. The remainder of this section provides a discussion of database reverse engineering and related research before presenting the innovations of the research. In Section 2, the extended Entity-Relationship model is reviewed. Section 3 provides an overview of relational databases and inclusion dependencies. Classification schemes for relations and attributes are also presented in this section. Sections 4 and 5 then detail the methodology and its extraction process. A summary and concluding remarks are found in Section 6.

1.1. Database reverse engineering

This research is strongly motivated by software reverse engineering concepts. Software reverse engineering deals with the problem of comprehending an existing system and recovering corresponding design specifications [32]. Thus, reverse engineering is part of the maintenance process that obtains a sufficient understanding of the system and its application domain to allow appropriate changes to be made [10]. It is regularly applied to improve software systems. Figure 1 illustrates the concepts of database reverse engineering.

1.1.1 Overview and definition

The overall object of this research is to develop a database reverse engineering methodology which can obtain, at a high level of automation, a 'good' EER model that corresponds to the design specifications of an existing relational database. As a result, we need an understanding of the relationship between database design specifications stated in an EER



Application Domain (Real World Objects)
Validation (DBA, Developers, End-Users)
Mapping
Semantic Degradations
Database Design and Maintenance
Conceptual Model (Modelling Structures)
Mapping
Semantic Degradations
Reverse Engineering (Interpretations)
Database System (Logical Data Model)

Fig. 1. Database reverse engineering.

model and the database implementation. Obtaining an EER model corresponds to the 'Reverse Engineering' arrow of Fig. 1, which provides interpretations (meanings) of the existing database structures. More specifically, a good EER model is one that would be mapped by generally accepted database design principles into the existing relational database (i.e. the inverse of the Reverse Engineering arrow in Fig. 1). This notion is detailed in Section 5.

Note that we do not assume that the database was originally designed using an EER approach. Indeed, a common motivation for performing reverse engineering is that the original design process and subsequent maintenance (the left-hand side of Fig. 1) were not well documented and not carried out in a methodical manner. Thus the question of what the original design specifications were is unanswerable, and often uninteresting, since good design methods cannot be assumed to have been followed. Additionally, the presence of the data populating the database can provide information that was not available at design time, so that we should expect an EER model which incorporates that information, together with the direct results of the original design decisions (i.e. the data schema), to be superior to the original design.

Once it is created, the EER model should be evaluated in two ways. First, the EER model can be evaluated in a syntactic sense. For example, the presence of an entity type which does not participate in any relationship may be a sign of a bad design. This kind of checking may be automatable and is the subject of future research. Second, the EER model must be evaluated as a representation of the portion of the real world it is supposed to model. This judgement corresponds to the 'Validation' arrow in Fig. 1, and can only be made by humans who understand the domain in question. Verification and validation issues are further discussed in Section 5.6.

Database reverse engineering is defined, in this research, as the process of examining an existing database system to:

(1) Identify the database's contents (e.g. relations and attributes in a relational database) and their interrelationships.
(2) Recover domain semantics which are not represented in the subject system explicitly (e.g. keys, cardinality ratios for relationships, etc.). The term 'domain semantics' refers to information about the application domain, which should be captured during the requirements specification phase of database design.
(3) Discover domain semantics which are either difficult to obtain or unobtainable by a forward engineering process.
(4) Propose possible design specifications that could lead to the existing system, based on the obtained semantics.
(5) Represent the results in a conceptual model (e.g. an Entity-Relationship model) that can provide a better interpretation of data with respect to the application domain.

1.1.2 Motivations and Advantages

There are various reasons for applying reverse engineering to existing databases as outlined below.

Semantic degradations During the design and maintenance of a database, some domain semantics might not be captured, or else are captured but removed due to representation and performance limitations of the database system [26, 17]. Therefore, design specifications of an existing database cannot be easily recognized by referring to information provided by the target database management system (DBMS). For example, it is often difficult to determine what kind of object is represented by a particular relation in a relational database. Database administrators, however, need design specifications to determine whether an existing database can accurately reflect the application domain and, if not, perform the logical and physical design of the database again. Also, a conceptual model can help users understand the database and, thus, retrieve the required data correctly [23].

Maintenance and redesign It is essential to obtain a conceptual model in order to develop a new data schema for an existing database. It is much easier and more accurate to create a new design by modifying an old one, rather than starting from scratch [1].

Integration of databases It may be necessary to integrate several databases. Such integration is best performed at the conceptual model level [2, 25]. Since conceptual models will often not exist for the databases in question, it will be necessary to reverse engineer them to arrive at the desired models [3].

Change of data models A conceptual schema will be needed as the intermediate design, in order to convert an existing database into a different data model (for example, a relational database into an object-oriented database).

Overall, the application of database reverse engineering technology has many advantages, such as the ability to leverage assets, reduce maintenance costs, facilitate technology transition, provide reliability to system upgrades leading to user satisfaction, etc. [32]. There are many tools available for software reverse engineering [11]. In addition, second-generation CASE systems incorporate reverse-engineering facilities that aid in modernizing existing databases [3]. Thus both researchers and the marketplace have recognized the importance of such methods.

1.2. Related research

Several researchers have provided some means for translating a relational data schema into an Entity-Relationship (ER) model, (e.g. [14, 13, 24, 19, 20, 22, 18]). Dumpala and Arora [14] report on the first research that focuses on the translation of the three traditional data models (hierarchical, network and relational) into the ER model. Davis and Arora [13] present two methods for translating the relational database into the original ER model. Navathe and Awong [24] provide a very detailed classification of relations and attributes. The classified relations and attributes are then translated into entity types, relationship types and categories of the Entity-Category-Relationship model [15]. In Navathe and Awong's method, inclusion dependencies are not considered. They hope that, by doing so, their method can be easily understood by practitioners who are likely to use it to construct a working tool. Markowitz and Makowsky [22] provide an algorithm to generate EER schemas from relational schemas which must consist of relation-schemes, key dependencies, and key-based inclusion dependencies. Ji et al. [18] present an algorithm which automatically converts relational schemas into Nested Entity-Relationship schemas. They claim that the resultant Nested Entity-Relationship schema provides both a natural and theoretically sound description of the relational database.

Johannesson and Kalman [19] appear to present the most complete methodology for translating a relational data schema into an EER model. However, in their approach, the user must specify all keys and inclusion dependencies, and then relations are classified based on how the primary key of a relation appears in inclusion dependencies. They also point out two common problems of research in the reverse engineering of relational databases. First, previous work seems to overestimate the possibility of automating the reverse engineering process. Some methods generate a conceptual model almost without any interaction with the user. Interaction with a knowledgeable person (the user) is imperative, though, to capture the essential domain semantics [17]. The user can be a database administrator, a database designer or a system analyst, or even the end-user. Second, the results generated by the above methods are often closely tied to the existing data schema and, therefore, may become just graphical means of representing the actual physical implementation of the database. Nevertheless, no research or tools for database reverse engineering have ever considered how to obtain domain semantics, such as inclusion dependencies and integrity constraints, by analyzing the extension (data instances) of an existing database.

1.3. Innovations

This research extends previous work in various ways as outlined below.

Data instances, heuristics, semantics The methodology uses, not only data schema and extraction rules, but also data instances and heuristics. Extraction rules, which describe how an EER model is obtained from a relational database system, are explicit enough to be encoded into a knowledge-based system.

Since semantic degradations occur during the design and maintenance of a database, this methodology identifies what kinds of domain semantics must be obtained from sources other than the target DBMS in order to generate the EER model. For example, a person who is knowledgeable in the application domain must be asked to provide the necessary semantics.

Inclusion dependencies The majority of the information that is necessary for the identification of relationship types is contained in inclusion dependencies [7, 19]. This methodology is the first to discuss how to discover inclusion dependencies. First, it formulates possible inclusion dependencies by using a set of heuristics to analyze the data schema. The heuristics used formulate only key-based inclusion dependencies in order to avoid generating many inappropriate dependencies. Additional inclusion dependencies could be provided manually. Then, data instances are analyzed to verify that they exist.

Classification schemes Exhaustive classification schemes are used to classify relations and attributes. The classification of relations and attributes allows them to be converted into corresponding EER modelling structures in a systematic fashion. This paper appears to be the first to present a thorough justification for the transformations applied to the existing database to produce the resulting EER model.

2. Extended Entity-Relationship model

The most widely used approach to conceptual modelling is the Entity-Relationship (ER) model [29], originally proposed by Chen [8]. Its fundamental modelling constructs are entities, relationships, and their associated attributes.

2.1. Attributes

Attributes represent descriptive properties of entities and relationships. Some attributes may be identifiers (keys). The identifiers form the set of candidate keys for an entity type from which one is designated as the primary key. Since a relationship type relates to several entity types, the primary key of a relationship type can be formed by the primary keys of its participating entity types [8]. Other attributes are descriptors, which describe properties of an entity or a relationship occurrence.

2.2. Entities

Entities of the same kind are collected into entity types, e.g. *Employee*, and *Customer*. Each entity type is described by a name and a set of attributes. Entities from the same entity type have common attributes. Entities can be distinguished into two types: strong entities and weak entities, based upon the *strength* of their identifying attributes [29]. Strong entities have internal identifiers that uniquely determine the existence of entity occurrences.

Weak entities cannot be uniquely identified by the values of their own attributes [8]. They derive their unique identification from the identifying attributes of other entities that are called *identifying owners*. The relationship that relates a weak entity to its identifying owner(s) is called the *identifying relationship*. A weak entity type may have an identifying relationship type of degree higher than two. In this case, a weak entity type can have more than one owner entity type [16]. In general, any number of levels of weak entities can be defined; that is, an owner entity may itself be a weak entity type [8].

2.3. Relationships

Relationships represent associations among entities [29]. *Employees work-for Departments*, for example, is an association between employees and departments. There must be at least two entity occurrences participating in each occurrence of a relationship. The entities that take part in a relationship are called the participating entities of the relationship [19]. Similar to entity types, relationships of the same kind are grouped into a relationship type. The following properties are usually specified for the relationship types.

Degree The degree of a relationship type is the number of participating entity types [16]. An n-ary relationship type denotes a relationship of degree n. Although relationships can be of any degree, the ones that occur most commonly are binary ($n = 2$) [29, 16].

Cardinalities The cardinality constraint specifies the number of relationship instances in which an entity can participate. The cardinality can be represented by its lower and upper bounds, called *Min* and *Max*, respectively [30]. Consider a binary relationship, *Managers manage Departments*. The *Min/Max* values of the cardinality indicate the minimum and maximum occurrences of entity type *Manager* that can occur for each occurrence of the entity type *Department*, and vice versa. If each occurrence of *Manager* can have exactly one corresponding occurrence of *Department*, then the *Min/Max* cardinalities for *Manager* are (1,1). On the other hand, if each occurrence of *Department* can have many occurrences of *Manager*, then the *Min/Max* cardinalities for *Department* are (1,N), where N denotes 'many'.

There are three basic *cardinality ratios* for binary relationship types: 1:1 (one to one), 1:N (one to many), and N:M (many to many) [21, 29, 16]. The appropriate cardinality ratio for a relationship type depends on the real world situation that the relationship represents.

Relationship attributes Similar to entity types, relationship types can have their own attributes. Consider the binary relationship type, *Employee work-for Departments*. If we want to record the start date that an employee works at a particular department, then 'start-date' becomes an attribute for the relationship type *work-for*. It is neither an attribute of *Employee* nor an attribute of *Department*, because its meaning depends on both the participating entities.

2.4. Data abstractions

Using the original ER model as a conceptual model, however, has proved difficult because of the inadequacy of the initial modelling constructs. The major contribution of the extended ER models is the addition of data abstractions such as aggregation, generalization and specialization [29, 16].

Aggregation Aggregation is a form of abstraction in which an association between objects is regarded as a higher level object, called a composite object [27]. A limitation of the original ER model is that a relationship type cannot be considered as an entity type, and so cannot itself participate in further relationships [12, 21]. The solution to this problem is to use the aggregation abstraction [21]. For example, a relationship type can be abstracted as an aggregation that groups its participating entity types as an object (composite entity type) in its own right. Composite entity types can represent complicated domain semantics better than the original ER entity type.

Generalization Generalization identifies that an entity type is the union of a set of non-overlapping *specific* entity types. If each occurrence of a *generic* entity type, G, appears in one and only one of its *specific* entity types, then a generalization hierarchy exists [29]. For example, if A *is-a* G and C *is-a* G hold and each occurrence of G is also an occurrence of either A or C but not both, then there is a generalization hierarchy between G with A and C.

Specialization has been defined as the inverse of generalization where one entity type is a simple subclass of another [30]. If each occurrence of an entity type A is also an occurrence of an entity type B, then there exists a subtype/supertype relationship between the entity types A and B, represented as A *is-a* B. A is called the *specific* entity type and B is called the *generic* entity type. For example, *Manager* is a specialization of *Employee*. These data abstractions are employed to represent the rich semantics of an application domain.

3. Relational database

In relational databases, the relation is the unique building structure. A relation is formally defined as an (unordered) set of unordered n-tuples [16]. An n-tuple is merely a set of n values used to represent an occurrence of some object in the application domain.

Inclusion dependencies formalize interrelational constraints among attributes across relations [7]. For example, referential integrity constraints cannot be specified as functional dependencies because they relate attributes across relations. A referential integrity constraint states that a tuple in one relation that refers to another relation must refer to an existing tuple in that relation [16]. Inclusion dependencies are also used to represent the constraint between keys of two relations which represent entity types that participate in an *is-a* relationship.

An inclusion dependency is denoted as $A.X \ll B.Y$, where A and B are relations, X is an attribute or a set of attributes of A, and Y is an attribute or a set of attributes of B. X and Y must have the same number of attributes. This inclusion dependency states that the set of values appearing in $A.X$ must be a subset of the set of values appearing in $B.Y$.

3.1. Entity and Relationship Representation

There are several methodologies for translating EER structures into relational models [31, 29, 28]. The translation of an EER model into a corresponding relational data schema involves representing both entity and relationship types by relations. Table 1 summarizes generally accepted translation rules for doing so. As it will be discussed in Sections 4 and 5, Table 1 forms the basis for selecting the EER interpretation of a given relational structure in a database which is being reverse engineered. The terminology used for relations and attributes is defined below. It extends the work of Dumpala and Arora [14] and Navathe and Awong [24][1].

3.2. Relations

A relation is classified mainly based upon the properties of its primary key; that is, the comparison of its primary key with keys of other relations. A sample of a relational data schema which is used throughout this paper is shown in Fig. 2[2].

[1] Navathe and Awong [24] call strong and weak entity relations Type 1 and Type 2 primary relations, respectively, and regular and specific relationship relations Type 1 and Type 2 secondary relations, respectively.
[2] The names of relations and primary keys are shown in capital letters.

Table 1
Relational representation of EER structures

EER STRUCTURE	RELATIONAL REPRESENTATION
Strong Entity Type.	Construct a strong entity relation consisting of key and non-key attributes of the corresponding entity type.
Weak Entity Type.	Construct a weak entity relation whose key attributes contain its identifying owner's keys, which implicitly represent the identifying relationship, plus its own key attributes (dangling key attributes).
Unary Relationship.	If it is a many-to-many relationship: • construct a new relationship relation. In all other cases: • make the key of the entity type a foreign key in the corresponding entity relation. *Key attributes need to be qualified with their entity names.*
Binary Relationship: A (1,1) and B (1,1).	If A and B have the same key, X: • specify referential integrity constraints, $A.[X] \ll B.[X]$ and $B.[X] \ll A.[X]$. If A and B have different keys: • make the key of A a foreign key of B; make the key of B a foreign key of A; or both.
Binary Relationship: only one of A or B has *Min/Max* values (1,1) (e.g., *is-a* relationship).	If A and B have the same key, X: • specify referential integrity constraints, either $A.[X] \ll B.[X]$ or $B.[X] \ll A.[X]$. If A and B have different keys: • make the key of the entity type with non-(1,1) values a foreign key in the resulting relation of the entity type with the (1,1) values.
Binary Relationship (all other cases) and N-ary Relationship (where $n > 2$).	Construct a new relationship relation whose primary key is either: • the concatenation of the keys of the participating entity types (a relationship relation), or • its own key (a strong entity relation).
Generalization Hierarchy.	If the *specific* entity types have the same key as the *generic* entity type: • specify integrity constraints. If the *specific* entity types have different keys: • add the key of the *generic* entity type as a foreign key in the resulting relations for the *specific* entity types, and specify integrity constraints.
Relationship between an entity type and a relationship type.	If the entity type has *Min/Max* values (1,1): • make the key of the entity type a foreign key in the resulting relationship relation. If the entity type has *Min/Max* values (0,1): • make the key of the entity type a foreign key in the resulting relationship relation, or • construct a new relation. Otherwise, construct a new relation. The primary key for the new relation can be either: • the concatenation of keys of the participating entity types (a relationship relation), or • its own key (a strong entity relation).

PERSON:	[SSN, name, address]
EMPLOYEE:	[SSN, name, salary, hired-date]
MANAGER:	[SSN, rank, promotion-date, deptno]
CUSTOMER:	[SSN, custid, name, credit]
DEPARTMENT:	[DEPTNO, dept-name, location]
PRODUCT:	[PRODID, description]
PRICE:	[PRODID, minprice, maxprice]
ORDER:	[ORDID, order-date, prodid, custid]
CARRIER:	[CARRIER-ID, name, address]
PROJECT:	[PROJNAME, DEPTNO, budget]
WORK-FOR:	[SSN, DEPTNO, start-date]
CAN-PRODUCE:	[DEPTNO, PRODID, unit-cost]
SHIPMENT:	[PACK#, ORDID, ship-date, carrier-id]

Fig. 2. Sample of a relational data schema.

Strong entity relation A strong entity relation is a relation whose primary key does not properly contain a key of any other relation. In Fig. 2, EMPLOYEE, MANAGER, and DEPARTMENT are examples of strong entity relations. WORK-FOR is not a strong entity relation because its primary key, [SSN, DEPTNO], contains the primary keys of EMPLOYEE or MANAGER, and DEPARTMENT.

Weak entity relation A weak entity relation is a relation that meets all of the following requirements:
(1) A proper subset of its primary key, called K1, contains the keys of entity relations (strong and/or weak).
(2) The remaining attributes of the primary key, called K2, do not contain a key of any other relation.
(3) This relation represents an entity type whose identifier must properly contain identifier attributes of other entity types (i.e. K1). This requirement must be confirmed by a knowledgeable user.

Consider PROJECT in Fig. 2. Its primary key contains the primary key, DEPTNO, of DEPARTMENT which is a strong entity relation. If PROJNAME is not a key of any other relation, and PROJECT represents an entity type, then PROJECT is a weak entity relation.

Regular relationship relation A regular relationship relation is a relation whose primary key is formed *fully* by the concatenation of keys of entity relations (strong and/or weak). For example, the relation CAN-PRODUCE in Fig. 2 is a regular relationship relation, because its primary key, [DEPTNO, PRODID], is formed *fully* by the concatenation of the primary keys of DEPARTMENT, and PRODUCT or PRICE.

Specific relationship relation A specific relationship relation is a relation whose primary key is *partially* formed by the concatenation of keys of entity relations (strong and/or weak), and it can not be treated as a weak entity relation. Consider SHIPMENT in Fig. 2. PACK# is not a key of any entity relation. If SHIPMENT cannot satisfy the third requirement for the weak entity relations, then SHIPMENT is a specific relationship relation.

3.3. Attributes

Attributes are first classified depending upon whether they are part of a relation's primary key. Then, depending on the type of relation, attributes of the primary key are divided into three categories. Non-primary-key attributes are classified into two. Non-primary-key attributes may contain candidate keys. The classification of attributes is summarized in Table 2.

Primary Key Attribute (PKA) Primary key attributes are:
(1) attributes of strong entity relations' primary keys,
(2) attributes of weak entity relations' primary key which appear as keys for some entity relations, and
(3) attributes of relationship relations' primary keys which are also keys of other relations.

Dangling Key Attribute (DKA) The attributes of the primary key of a weak entity relation which do not appear as key attributes of any other relations, are dangling key attributes.

General Key Attribute (GKA) The attributes of the primary key of a specific relationship relation which do not appear as key attributes of any other relations, are general key attributes.

Foreign Key Attribute (FKA) If a subset of non-primary-key attributes of a given relation appear as a key of another entity relation (strong or weak), then they are foreign key attributes of the given relation.

Non Key Attribute (NKA) The remaining attributes are non key attributes.
Consider the following relations which were classified previously:

```
DEPARTMENT: [DEPTNO, dept-name, location]         (strong)
PROJECT:    [PROJNAME, DEPTNO, budget]            (weak)
MANAGER:    [SSN, rank, promotion-date, deptno]   (strong)
WORK-FOR:   [SSN, DEPTNO, start-date]             (regular)
SHIPMENT:   [PACK#, ORDID, ship-date, carrier-id] (specific)
```

Table 2
Classification of relations' attributes (NA: Not Applicable)

Relation Type	PKA	DKA	GKA	FKA	NKA
STRONG	Required	NA	NA	Optional	Optional
WEAK	Required	Required	NA	Optional	Optional
REGULAR	Required	NA	NA	Optional	Optional
SPECIFIC	Required	NA	Required	Optional	Optional

DEPTNO in DEPARTMENT and [SSN, DEPTNO] in WORK-FOR are examples of primary key attributes. PROJNAME is classified as a dangling key attribute because PROJECT is a weak entity relation. PACK# is classified as a general key attribute because SHIPMENT is a specific relationship relation. The attribute 'deptno' in MANAGER is a foreign key attribute because it appears as the primary key of DEPARTMENT. The attributes 'dept-name' and 'location' in DEPARTMENT are examples of non key attributes.

The preceding classification schemes for relations and attributes can both be shown to be mutually exclusive and exhaustive.

4. Methodology

Table 1, together with the classification schemes presented in Section 3, forms the basis for carrying out the reverse engineering of an existing database. Since the design principles summarized there are generally accepted as leading to a good relational model given an EER model, reverse engineering proceeds by determining the type of each relational construct (e.g. strong entity relation in Section 3) in the database being reverse engineered and then asking 'what kind of EER structure would give rise to this?' The best answer to this question comes from Table 1 by looking at each EER structure which could give rise to the relational structure, and selecting the most likely alternative. This is summarized in Table 3, and the rules are discussed at length in Section 5.

More formally, the methodology divides the reverse engineering of relational databases into the following four steps:

(1) *Decomposition*: Decompose input relations into at least third normal form (3NF) relations
(2) *Classification*: Classify relations and attributes
(3) *Generation*: Generate inclusion dependencies
(4) *Identification*: Identify modelling structures of the EER model.

Figure 3 shows these steps, and the input, output and functions for each. There are four main aspects of the methodology each of which is discussed in detail below:

- Assumptions
- Required information for the extraction process
- The extraction process
- Verification and validation of the extraction process.

The extraction process performs reverse engineering of relational databases by recovering domain semantics from an existing database and representing the results as an EER model. Section 5 details the extraction process and its verification and validation.

4.1. Assumptions

Three major assumptions are made about the characteristics of the input databases[3].

[3] Future research will attempt to relax some of these assumptions.

Table 3
Identification of EER constructs by classified relations and attributes (This table is the inverse of the EER to relational mapping shown in Table 1)

Relation	Attribute	Modelling Constructs in the EER Model
STRONG		• **Identifies a strong entity type**, or • **Identifies a relationship type with its own key**
	PKA	Key for the strong entity type
	FKA	Identifies a binary relationship
	NKA	Descriptive attribute for a relationship or an entity type
WEAK		**Identifies a weak entity type**
	PKA	Relates a weak entity type with its identifying owner(s)
	DKA	Key for the weak entity type
	FKA	Identifies a binary relationship
	NKA	Descriptive attribute for a relationship or an entity type
REGULAR		• **Identifies an n-ary relationship type**, or • **Identifies a relationship between an entity type and a relationship type**, if $n > 2$
	PKA	Determine participating entity types
	FKA	Identifies a binary relationship between an entity and a relationship
	NKA	Descriptive attribute for a relationship type
SPECIFIC		• **Identifies an n-ary relationship type**, or • **Identifies a relationship between an entity type and a relationship type**, if $n > 2$
	PKA	Determines participating entity types
	GKA	Identifies new strong entity type(s)
	FKA	Identifies a binary relationship between an entity and a relationship
	NKA	Descriptive attribute for a relationship type

3NF relations Previous research has studied how to infer functional dependencies by analyzing data instances in the existing database [5], and there is at least one commercially available tool, DB Designer, by Cadre Technologies Inc., which analyzes real data using statistical analysis to determine automatically functional dependencies. Some standard algorithm (e.g. Bernstein [4]) can then be used to decompose these relations into ones that satisfy the 3NF requirement [12].

Since Step 1 of Fig. 3 is well understood, the methodology first assumes that the relations of the input database are in at least 3NF[4] (i.e. Step 1 of Fig. 3 has been completed). This

[4] If there is a non 3NF relation in the input database, then this relation will be converted into an aggregation which represents more than one entity type or a mixture of relationship types and entity types. It can be decomposed into detailed modelling structures manually.

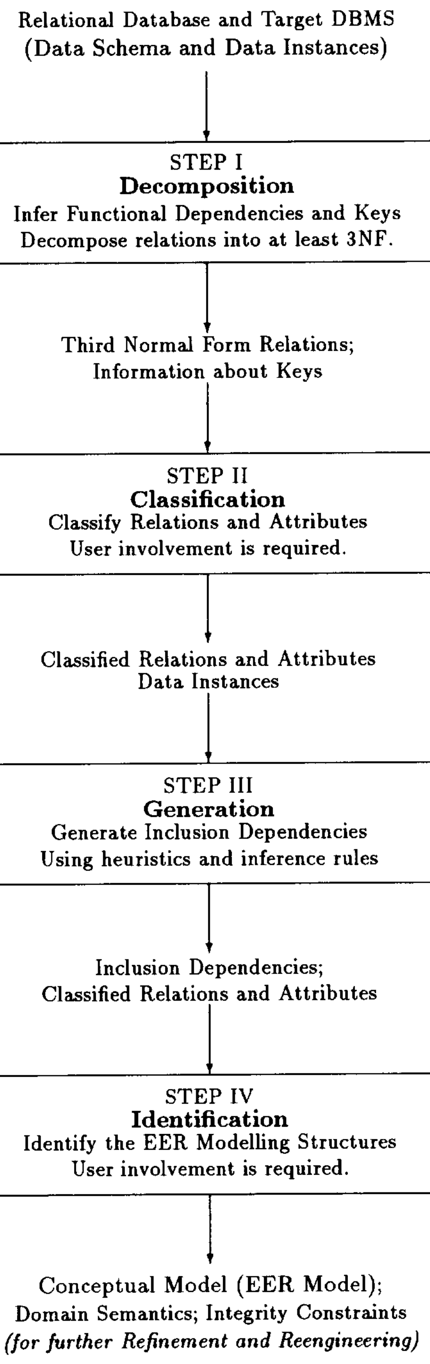

Relational Database and Target DBMS
(Data Schema and Data Instances)
STEP I
Decomposition
Infer Functional Dependencies and Keys
Decompose relations into at least 3NF.
Third Normal Form Relations;
Information about Keys
STEP II
Classification
Classify Relations and Attributes
User involvement is required.
Classified Relations and Attributes
Data Instances
STEP III
Generation
Generate Inclusion Dependencies
Using heuristics and inference rules
Inclusion Dependencies;
Classified Relations and Attributes
STEP IV
Identification
Identify the EER Modelling Structures
User involvement is required.
Conceptual Model (EER Model);
Domain Semantics; Integrity Constraints
(for further Refinement and Reengineering)

Fig. 3. Steps in the reverse engineering of relational databases.

assumption simplifies the extraction process because each relation will correspond to one entity type or one relationship type, rather than corresponding to more than one entity type or a mixture of entity and relationship types [24].

Consistent naming of key attributes A consistent naming convention is applied to key attributes. That is, if key attributes have the same domain, then they have the same name throughout all relations. This convention must be followed in the process of database design; otherwise it is very difficult for database users to query data involving several relations. This allows:
(1) each relation and attribute in the input database to be classified into its proper category in Step 2 of Fig. 3, and
(2) possible key-based inclusion dependencies to be formulated in Step 3 of Fig. 3.

No error data in values of key attributes The input database does not contain any erroneous data instances in its key attributes. This allows all possible inclusion dependencies to be discovered in Step 3 of Fig. 3.

4.2. Required information for the extraction process

The extraction process must have the following types of information in order to function:

Data schema Initially, information about the data schema must be available. This includes relation names, attribute names, and primary keys. In general, information about the relation and attribute names can be obtained directly by querying the target DBMS. If the DBMS does not contain information about primary keys, the user must specify the primary key for each relation. Information about attributes' properties, such as whether they are uniquely indexed and have unique and non-null values, can be applied to reduce the number of possible attributes for the primary key of each relation.

In other methods (e.g. Navathe and Awong [24]; Johannesson and Kalman [19]), the user needs to specify all candidate keys for each relation and may need to substitute a candidate key for the current primary key. Using our methodology, in contrast, information about candidate keys is requested only when:
(1) there are ambiguities in the classification of relations, and
(2) the user specifies inclusion dependencies between non key attributes.

Data instances The extraction process analyzes data instances to:
(1) verify the proposed inclusion dependencies, and
(2) identify candidate keys.

Inclusion dependencies Inclusion dependencies are necessary for identifying:
(1) *inclusion* relationships,
(2) generalization hierarchies,
(3) owner entity types for weak entities, and
(4) participating entity types for relationships.

The generation of key-based inclusion dependencies is an automated process. In addition, the extraction process allows the user to specify inclusion dependencies between non key attributes.

Domain semantics obtained from the user Because of the limited semantic expressiveness of the current DBMSs, database reverse engineering cannot be a totally automated process. User involvement is necessary whenever some domain semantics cannot be inferred or there are ambiguities which cannot be solved mechanically.

Heuristics Certain types of heuristics are used to:
(1) propose possible primary key attributes for each relation,
(2) formulate possible key-based inclusion dependencies, and
(3) specify default cardinality ratios for binary relationship types.

Extraction rules Extraction rules are used to:
(1) classify relations and attributes (classification rules),
(2) generate inclusion dependencies (inference rules), and
(3) convert the classified relations and attributes into corresponding modelling structures in the EER model (identification and assignment rules).
Each of these rules is discussed in the following section.

5. The extraction process

The extraction process consists of the following four steps:
(1) Classification of relations and attributes (Step 2 of Fig. 3)
(2) Generation of inclusion dependencies (Step 3 of Fig. 3)
(3) Identification of entity and relationship types (Part of Step 4 of Fig. 3)
(4) Assignment of attributes (Part of Step 4 of Fig. 3).

5.1. Classification of relations and attributes

The rules embodied in Table 3 require that relations and attributes are classified before being mapped to appropriate EER constructs. The classification schemes used for this purpose are the two introduced in Section 3. The classification rules follow directly from the definitions of the relation and attribute types presented there, and therefore are not presented in detail here.

Information about primary keys is first used to classify both strong entity relations and those regular relationship relations that contain only primary keys of strong entity relations. Then, the remaining relations whose primary keys contain attributes that do not appear as a primary key of any other strong entity relation are classified.

Suppose that the primary key attributes which do not appear as primary key attributes in another relation do appear as the non-primary-key attributes in other strong entity relations and are verified as candidate key(s) by analysis of the data instances. Then this relation is classified as a regular relationship relation. For example:

```
PROJECT: [PROJNAME, DEPTNO, budget].
```

If PROJNAME is a candidate key for a strong entity relation, then PROJECT is classified as a regular relationship relation. This means that a strong entity relation has PROJNAME as a non key attribute which contains non-null and unique values of data instances. The information about candidate keys will be used in formulating inclusion dependencies.

If a proper subset of a relation's primary key attributes does not appear as a key in any other relation, then it is impossible to classify this relation to be either a weak entity relation or a specific relationship relation even by analysis of the data instances. In order to classify such a relation, the extraction process must refer to domain semantics which are captured in the design phase but not represented in the data schema. For example:

```
SHIPMENT: [PACK#, ORDID, ship-date, carrier-id].
```

Suppose that PACK# does not appear as a key of any other strong entity relation (i.e. SHIPMENT satisfies requirements 1 and 2 of the definition of the weak entity relation). Then the user must confirm whether SHIPMENT is a weak entity relation or a specific relationship relation. SHIPMENT is classified as a weak entity relation if it also satisfies the third requirement for the weak entity relation; otherwise, it is a specific relationship relation.

Attributes for each classified relation can then be classified without user assistance. Table 4 shows the classification of the relations and attributes of the sample data schema given in Fig. 2.

5.2. Generation of inclusion dependencies

Inclusion dependencies contain the majority of the information for the identification of relationship types. The extraction process detects possible inclusion dependencies by referring to classified relations and attributes, and rejects invalid ones by analyzing data instances. The

Table 4
Classification of relations and attributes

Relation	TYPE	PKA	DKA	GKA	FKA	NKA
PERSON	strong	[SSN]	-	-	-	[name, address]
EMPLOYEE	strong	[SSN]	-	-	-	[name, salary, hired-date]
MANAGER	strong	[SSN]	-	-	[deptno]	[rank, promotion-date]
CUSTOMER	strong	[SSN]	-	-	-	[custid, name, credit]
DEPARTMENT	strong	[DEPTNO]	-	-	-	[dept-name, location]
PRODUCT	strong	[PRODID]	-	-	-	[description]
PRICE	strong	[PRODID]	-	-	-	[minprice, maxprice]
ORDER	strong	[ORDID]	-	-	[prodid]	[order-date, custid]
CARRIER	strong	[CARRIER-ID]	-	-	-	[name, address]
PROJECT	weak	[DEPTNO]	[PROJNAME]	-	-	[budget]
WORK-FOR	regular	[SSN, DEPTNO]	-	-	-	[start-date]
CAN-PRODUCE	regular	[DEPTNO, PRODID]	-	-	-	[unit-cost]
SHIPMENT	specific	[ORDID]	-	[PACK#]	[carrier-id]	[ship-date]

heuristics and inference rules employed by the extraction process are described in detail below.

5.2.1 Heuristics for proposing possible inclusion dependencies

The first step is to formulate possible inclusion dependencies. In order to avoid formulating many inappropriate dependencies, the extraction process uses heuristics that only formulate inclusion dependencies between relations' key attributes. The heuristics are outlined below:

(1) **IF:** two strong entity relations, A and B, have the same key, X;
THEN: there may be an inclusion dependency between $A.X$ and $B.X$; that is, either $A.X \leqslant B.X$ or $B.X \leqslant A.X$.
JUSTIFICATION: The existence of subtype/supertype relationships between entity types result in such inclusion dependencies. Therefore, these inclusion dependencies are used to identify *inclusion* relationships.
Consider the following relations:

```
PERSON:    [SSN, name, address]                 (strong)
EMPLOYEE:  [SSN, name, salary, hired-date]      (strong)
MANAGER:   [SSN, rank, promotion-date, deptno]  (strong)
CUSTOMER:  [SSN, custid, name, credit]          (strong)
PRODUCT:   [PRODID, description]                (strong)
PRICE:     [PRODID, minprice, maxprice]         (strong)
```

The following pairs of key attributes are considered:

(PERSON.[SSN], EMPLOYEE.[SSN])
(PERSON.[SSN], MANAGER.[SSN])
(PERSON.[SSN], CUSTOMER.[SSN])
(EMPLOYEE.[SSN], MANAGER.[SSN])
(EMPLOYEE.[SSN], CUSTOMER.[SSN])
(MANAGER.[SSN], CUSTOMER.[SSN])
(PRODUCT.[PRODID], PRICE.[PRODID])

(2) **IF:** the key, X, of an entity relation (strong or weak), A, appears as a foreign key, x, of another relation (entity or relationship), B;
THEN: there may be an inclusion dependency between $B.x$ and $A.X$, denoted by $B.x \leqslant A.X$.
JUSTIFICATION: In Table 1, the cases where binary relationships and relationships between entity and relationship types are represented by foreign keys, result in such inclusion dependencies.
Consider the following relations:

```
MANAGER:     [SSN, rank, promotion-date, deptno]    (strong)
DEPARTMENT:  [DEPTNO, dept-name, location]          (strong)
SHIPMENT:    [PACK#, ORDID, ship-date, carrier-id]  (specific)
CARRIER:     [CARRIER-ID, name, address]            (strong)
```

The following pairs of attributes are considered:

(MANAGER.[deptno], DEPARTMENT.[DEPTNO])
(SHIPMENT.[carrier-id], CARRIER.[CARRIER-ID])

(3) **IF:** The primary key attributes, *X*, of a relationship relation (regular of specific) or a weak entity relation, *S*, appear as a key of an entity relation, *A*;
THEN: there may be an inclusion dependency between *S.X* and *A.X*, denoted by $S.X \ll A.X$.
JUSTIFICATION: The presence of relationships represented by relationship relations or weak entities will result in this type of inclusion dependency. These dependencies are used to determine the owner entity types for weak entities and participating entity types for relationships.
Consider the following relations.

```
WORK-FOR:   [SSN, DEPTNO, start-date]              (regular)
PROJECT:    [PROJNAME, DEPTNO, budget]             (weak)
EMPLOYEE:   [SSN, name, salary, hired-date]        (strong)
MANAGER:    [SSN, rank, promotion-date, deptno]    (strong)
DEPARTMENT: [DEPTNO, dept-name, location]          (strong)
```

The following pairs of key attributes are considered:

(WORK-FOR.[SSN], EMPLOYEE.[SSN])
(WORK-FOR.[SSN], MANAGER.[SSN])
(WORK-FOR.[DEPTNO], DEPARTMENT.[DEPTNO])
(PROJECT.[DEPTNO], DEPARTMENT.[DEPTNO])

The above heuristics are not sufficient to detect all possible key-based inclusion dependencies. Suppose that a candidate key of a strong entity relation, which is not identified in the classification of relations, appears as a non key attribute(s) in another relation, then a key-based inclusion dependency may exist. For example, in Fig. 2, the attribute 'custid' in CUSTOMER is a candidate key and it also appears as a non key attribute in ORDER. The extraction process, therefore, allows the user to specify additional inclusion dependencies between non key attributes. For example, the user specifies an inclusion dependency, ORDER.[custid] $\ll$ CUSTOMER.[custid]. Then, the attribute 'custid' in CUSTOMER must be verified as a candidate key by analysis of the data instances.

5.2.2 Rejection of invalid inclusion dependencies

Each of the proposed inclusion dependencies is subject to further analysis. Two rules are used to reject invalid inclusion dependencies. Let (*A.X*, *B.X*) be a pair of attributes. *Value*(*A.X*) denotes the set of values of *A.X*, and *Value*(*B.X*) denotes the set of values of *B.X*.

IF: *Value*($B.X$) $\neq \emptyset$ and *Value*($B.X$) $\subseteq$ *Value*($A.X$);
THEN: $B.X \ll A.X$ cannot be rejected.
IF: *Value*($A.X$) $\neq \emptyset$ and *Value*($A.X$) $\subseteq$ *Value*($B.X$);
THEN: $A.X \ll B.X$ cannot be rejected.
JUSTIFICATION: These rules are based on the definition of inclusion dependencies.

For each proposed inclusion dependency, the extraction process must compare the sets of values appearing in $A.X$ and $B.X$. For example, the relations EMPLOYEE and CUSTOMER could have the same primary key, SSN. It is impossible, however, for there to be a subset relationship between EMPLOYEE.[SSN] and CUSTOMER.[SSN], so that the proposed inclusion dependency between EMPLOYEE.[SSN] and CUSTOMER.[SSN] is rejected. Consider another example where there is a proposed inclusion dependency between MANAGER.[SSN] and EMPLOYEE.[SSN]. If the value set of MANAGER.[SSN] is a proper subset of the value set of EMPLOYEE.[SSN], then MANAGER.[SSN] ≪ EMPLOYEE.[SSN] cannot be rejected.

If a relation's foreign key attributes do not appear in any valid inclusion dependencies, then they must be reclassified as non key attributes.

5.2.3 Removal of redundant inclusion dependencies

Redundant inclusion dependencies can be detected by the inference rules of inclusion dependencies (projection and transitivity) [7, 16]. That is:

IF: $A.X \ll B.X$ and $B.Y \ll C.Y$ hold, and Y is a subset of X;
THEN: the inclusion dependency $A.Y \ll C.Y$ is redundant.
JUSTIFICATION: The inference rules for inclusion dependencies.

There are two reasons for eliminating redundant inclusion dependencies. First, a redundant inclusion dependency can lead to a redundant *is-a* relationship [19]. Suppose, for example, that the following inclusion dependencies are discovered.

```
MANAGER.[SSN]  ≪EMPLOYEE.[SSN]
EMPLOYEE.[SSN]≪PERSON.[SSN]
MANAGER.[SSN]  ≪PERSON.[SSN]
```

The last inclusion dependency is redundant because it can be inferred from the first two. If it were not eliminated, then an *is-a* relationship from *Manager* to *Person* would be identified. This relationship is obviously redundant because there exist *is-a* relationships from *Manager* to *Employee* and from *Employee* to *Person*.

Second, a redundant inclusion dependency can lead to identifying the wrong participating entity type for a relationship type. Consider the following inclusion dependencies.

```
WORK-FOR.[DEPTNO]≪DEPARTMENT.[DEPTNO]
WORK-FOR.[SSN]    ≪EMPLOYEE.[SSN]
EMPLOYEE.[SSN]    ≪PERSON.[SSN]
WORK-FOR.[SSN]    ≪PERSON.[SSN]
```

The last inclusion dependency is redundant. If it is used to infer the participating entity types for the relationship type, *work-for*, identified by WORK-FOR, then the entity type, *Person*, identified by PERSON becomes a participating entity type. However, *Person* should not be inferred as the participating entity type of *work-for* based upon the *is-a* relationship from *Employee* to *Person* and a binary relationship between *Employee* and *Department*.

5.3. Identification of entity types

Entity types must be identified first so relationship types among them can be identified thereafter. The existence of an entity relation indicates the presence of an entity type. Strong entity relations identify strong entity types, whereas weak entity relations identify weak entity types and corresponding identifying relationships. The name of the entity relation is assigned to be that of the corresponding entity type. The identification rules are discussed in detail below.

5.3.1 Strong entities

IF: a relation is classified as a strong entity relation;
THEN: identify a strong entity type with the primary key of this relation as its key.
JUSTIFICATION: According to Table 3, a strong entity relation can be converted into one of two EER modelling structures. Since a relationship type with its own key can be considered as an entity type on its own, all strong entity relations are converted and represented as strong entity types in the EER model.

In the sample data schema, EMPLOYEE is a strong entity relation and is, therefore, converted into a strong entity type, *Employee*, which is identified by SSN. A strong entity type is represented as a rectangular box and its key attribute(s) is shown in an oval with its name(s) underlined as shown in Fig. 4.

5.3.2 Weak entities

IF: a relation is classified as a weak entity relation;
THEN: identify a weak entity type with the dangling key attribute(s) of this relation as its key.
JUSTIFICATION: In Table 3, a weak entity relation can only be converted into a weak entity type.

In addition, an identifying relationship between the weak entity type and its identifying owner(s) must be identified. The owner entity types are determined by referring to inclusion

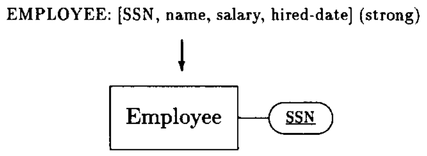


Fig. 4. Identification of a strong entity type.

dependencies. The following rule is used to identify the identifying relationship for a weak entity type.

IF:
(1) a weak entity type is identified by a weak entity relation, W,
(2) the primary key attribute(s), X, of W appears as a key of an entity relation (strong or weak), A, and
(3) there is an inclusion dependency, $W.X \ll A.X$;
THEN: the entity type identified by A is the owner entity type for the weak entity type identified by W.
IF: more than one entity relation satisfies the above conditions;
THEN: the user must determine the exact owner entity.
JUSTIFICATION: Conditions 1, 2 and 3 above are necessary conditions for the entity type identified by A to be an owner entity type of the weak entity type identified by W. If in addition A is unique in satisfying those conditions, then they are sufficient as well. If A is not unique, then further information which can only be supplied by the user is needed.

The usual cardinality ratio for a binary identifying relationship is 1:N. Similar to strong entity relations, the name of a weak entity relation is assigned to be the name for the weak entity type. The extraction process provides a constant name, 'DEPENDENT', for every identifying relationship. Consider PROJECT in Table 4. A weak entity type, *Project*, is identified with its own dangling key attribute, PROJNAME, as the key. *Project* has an identifying relationship with its owner entity type, *Department*, which is determined by the inclusion dependency: PROJECT.[DEPTNO] $\ll$ DEPARTMENT.[DEPTNO].

Figure 5 shows the representation of a weak entity type, *Project*, and its identifying relationship and owner in the EER diagram. A weak entity type is represented by a double rectangular box and its key attributes are shown in ovals with their names underlined. An arrow terminating on the double rectangular box shows that the identifying relationship has the dependent property. The source of the arrow shows the owner entity type on which the weak entity type depends.

5.4. Identification of relationship types

The extraction process identifies five kinds of relationships. These are:
(1) *inclusion* relationships,
(2) generalization hierarchies,

DEPARTMENT: [DEPTNO, dept-name, location] (strong)
PROJECT: [PROJNAME, DEPTNO, budget] (weak)
PROJECT.[DEPTNO] $\ll$ DEPARTMENT.[DEPTNO]

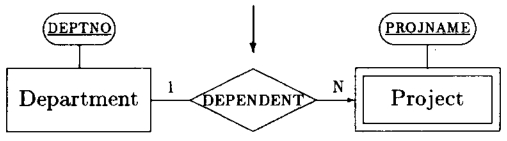


Fig. 5. Identification of a weak entity type.

(3) binary relationships identified by foreign keys of entity relations, and inclusion dependencies specified by the user,
(4) n-ary relationships identified by relationship relations, and
(5) aggregate relationships identified by foreign keys of relationship relations.

Primarily, relationship relations are converted into relationship types. Inclusion dependencies are necessary in determining participating entity types for each relationship type. For example, in Table 4, DEPTNO is a primary key attribute in the regular relationship relation, WORK-FOR, and it is also the primary key of the entity relation, DEPARTMENT. This is not a sufficient indication that the entity type identified by DEPARTMENT is a participating entity type for the relationship type identified by WORK-FOR; an inclusion dependency must also hold.

There are other kinds of relationships that cannot be identified from relationship relations, such as *is-a* relationships and binary relationships represented by foreign keys. Inclusion dependencies, however, contain crucial information for identifying them. Thus, the cross reference between key attributes and inclusion dependencies will determine how relationship types are identified.

5.4.1 Inclusion relationships

The extraction process uses inclusion dependencies to identify *inclusion* relationships. *Is-a* relationships are identified by the following rule:

IF:
(1) two strong entity relations, A and B, have the same key, X, and
(2) there is an inclusion dependency, $A.X \ll B.X$; between their keys;
THEN: identify an *is-a* relationship between the entity types that are identified by A and B respectively.
JUSTIFICATION: From Table 1, binary relationships having at least one (1,1) *Min/Max* values are the only EER structures leading to such relational structures. Thus an *is-a* relationship is the most appropriate interpretation of the relational structures which satisfy the above conditions.

Consider the strong entity relations, EMPLOYEE and MANAGER, and the inclusion dependency, MANAGER.[SSN] $\ll$ EMPLOYEE.[SSN]. The inclusion dependency suggests that an *is-a* relationship exists from *Manager* to *Employee* as shown in Fig. 6. The cardinality ratio for an *is-a* relationship is always 1:1. If two entity types have, not only the same key, but

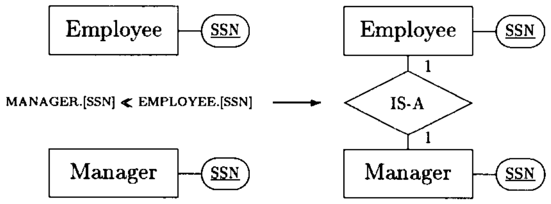


Fig. 6. Identification of an *is-a* relationship.

also the same set of data instances in their keys, then the user must specify the proper type of *inclusion* relationship between them, such as *A is-a B*, *A is-a-kind-of B*, *A is-part-of B*, *A has B*, etc. The rule for identifying *inclusion* relationships is:

IF:

(1) two strong entity relations, *A* and *B*, have the same key, *X*, and
(2) there are two inclusion dependencies, $A.X \ll B.X$ and $B.X \ll A.X$, between their keys;

THEN:

(a) identify an *inclusion* relationship between entity types identified by *A* and *B* respectively, and
(b) specify the proper type of inclusion relationship.

JUSTIFICATION: *A* and *B* having, not only the same key, but also the same set of values in their keys, can be best interpreted as the existence of an *inclusion* relationship. The *inclusion* relationship is represented as a binary relationship type in the EER model with a proper name.

Consider PRODUCT and PRICE in Table 4. Suppose that the following two inclusion dependencies hold:

```
PRODUCT.[PRODID]≪PRICE.[PRODID]
PRICE.[PRODID]  ≪PRODUCT.[PRODID]
```

PRODUCT and PRICE are strong entity relations. They have, not only the same primary key, but also the same set of data instances for their primary keys. In this case, the user must specify a proper type of *inclusion* relationship between entity types identified by them. For example a binary relationship type called 'HAS' is specified as shown in Fig. 7.

5.4.2 Generalization hierarchies

Generalization hierarchies are identified by the following rule:

IF:

(1) a *generic* entity type has more than one *specific* entity type (note that these are identified by *is-a* relationships),
(2) there is a complete covering of the *generic* entity type (every occurrence of the *generic* entity type appears in at least one of the *specific* entity types), and

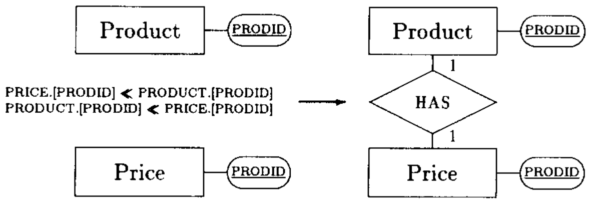

Product
PRODID
PRICE.[PRODID] ≪ PRODUCT.[PRODID]
PRODUCT.[PRODID] ≪ PRICE.[PRODID]
Price
PRODID
Product
PRODID
1
HAS
1
Price
PRODID

Fig. 7. Identification of an *inclusion* relationship.

(3) there is a non-overlapping partition of the *generic* entity type (every occurrence of the *generic* entity type appears in at most one of the *specific* entity types);

THEN: a generalization hierarchy exists between the *generic* entity type and its *specific* entity types.

JUSTIFICATION: Definition of a generalization hierarchy.

For example, if *Customer* and *Employee* are *specific* entity types related to the same *generic* entity type, *Person*, there may be a generalization hierarchy among them. In order to verify this, the extraction process must analyze data instances in order to satisfy two integrity constraints. First, if each instance of *Person* is an instance of either a *Customer*, or an *Employee*, then *Customer* and *Employee* form a complete covering of *Person*. Second, if an instance of *Person* is an instance of either *Customer*, or *Employee*, but not both, then *Customer* and *Employee* are non-overlapping subsets of *Person*.

5.4.3 Binary relationships

Foreign keys of entity relations and inclusion dependencies specified by the user are used to identify binary relationship types. First, a foreign key in an entity relation indicates that a binary relationship type exists. The identification rule is:

IF:

(1) a foreign key, x, of an entity relation (strong or weak), B, appears as a key, X, of another entity relation (strong or weak), A,

(2) there is an inclusion dependency between them, $B.x \ll A.X$;

THEN:

(a) identify a binary relationship type which has two participating entity types. One is identified by the relation containing the foreign key and the other is identified by the relation whose key is the same as the foreign key.

(b) specify a proper name for this relationship.

IF: more than one entity relation, A, satisfies the above conditions;

THEN: the user must determine the exact participating entity type.

JUSTIFICATION: From Table 1, foreign keys in entity relations are used to represent binary relationships between entity types. Thus, each foreign key of an entity relation is converted into a binary relationship type. Similar to the determination of owner entity types for the weak entity types case, conditions 1 and 2 above are necessary conditions for the entity type identified by A to be a participating entity type of the binary relationship type identified by the foreign key, x. If A is not unique in satisfying those conditions, then further information which can only be supplied by the user is needed.

Since the name of such relationships is normally not stored in the target DBMS, the user must specify them. The cardinality ratio for a binary relationship type identified by a foreign key is either 1:1 or 1:N. If the foreign key contains unique values then the cardinality ratio is 1:1.

Suppose there are entity types *Department* and *Manager*, and an inclusion dependency, MANAGER.[deptno] $\ll$ DEPARTMENT.[DEPTNO]. The extraction process identifies a

MANAGER: [SSN, rank, promotion-date, deptno] (strong)
DEPARTMENT: [DEPTNO, dept-name, location] (strong)
MANAGER.[deptno] ≪ DEPARTMENT.[DEPTNO]

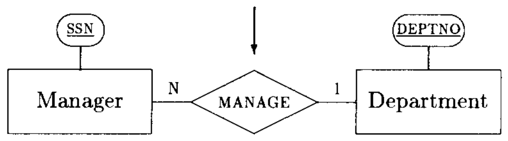


Fig. 8. A foreign key identifies a binary relationship.

binary relationship between the entity types *Manager* and *Department*, for which the user must provide a name; for example, 'MANAGE' as shown in Fig. 8. If the foreign key is also a candidate key, then the binary relation could be an *is-a* relationship. For example:

```
PERSON: [SSN, name, address]         (strong)
CUSTOMER: [CUSTID, ssn, name, credit] (strong)
CUSTOMER.[ssn]≪PERSON.[SSN]
```

In this case, the strong entity relation, CUSTOMER, has its primary key, CUSTID, instead of 'ssn' which is a foreign key. An *is-a* relationship between *Person* and *Customer* could be identified by the user as shown in Fig. 9. The extraction process must analyze data instances to verify that the non key attribute, 'ssn', is a candidate key for CUSTOMER.

Second, a valid inclusion dependency between non key attributes (one of which is a candidate key), specified by the user, indicates that a binary relationship type must be identified. Similar to the foreign key case, the user must provide a proper name for this relationship. The identification rule is:

IF: there is an inclusion dependency between non key attributes, $A.x \ll B.y$;
THEN:
(a) identify a binary relationship type consisting of two participating entity types identified by the relations A and B, and
(b) specify a proper name for this relationship.
JUSTIFICATION: Similar to the justification of the identification of binary relationships by foreign keys of entity relations.

PERSON: [SSN, name, address] (strong)
CUSTOMER: [CUSTID, ssn, name, credit] (strong)
CUSTOMER.[ssn] ≪ PERSON.[SSN]

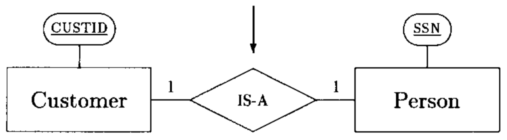


Fig. 9. A foreign key identifies an *is-a* relationship.

5.4.4 Relationships identified by relationship relations

A relationship relation indicates the presence of an association among entity types. For each relationship relation, the extraction process identifies an *n*-ary relationship type among the participating entity types whose keys form the primary key of this relation. The relationship relation's name is assigned to be the name of the corresponding relationship type. As previously mentioned, there are two types of relationship relations: regular and specific. The manner in which they are converted into relationship types is different. For each regular relationship relation, the extraction process identifies a relationship type among participating entity types that have already been identified by the extraction process. For each specific relationship relation, however, one or more entity types must be identified first. Then, a relationship type is identified among all participating entity types.

The participating entity types are determined by examining the primary keys and inclusion dependencies. The degree of a relationship type is determined by the number of participating entity types. The extraction process assigns an N:M cardinality ratio for a binary relationship type identified by the relationship relation. This methodology, however, does not specify cardinality ratios for relationship types of degree higher than two.

Regular relationship relations A regular relationship relation identifies an *n*-ary relationship type. The identification rule is:

> **IF:** there is a regular relationship relation;
> **THEN:** identify an *n*-ary relationship type among the entity types whose keys form the relation's primary key.
> **JUSTIFICATION:** In Table 3, a regular relationship relation can be converted into an *n*-ary relationship type or a relationship type between an entity type and a relationship type if its degree is higher than two. Since the latter situation is a special case of the former, all regular relationship relations are converted into *n*-ary relationship types.

The participating entity types for a relationship type identified from a regular relationship relation are determined by the following rule:

> **IF:**
> (1) a subset, X, of the primary key attributes of a regular relationship relation, R, appears as a key of an entity relation (strong or weak), A, and
> (2) there is an inclusion dependency between them, $R.X \ll A.X$;
> **THEN:** specify the entity type identified by the relation, A, as the participating entity type for the relationship type identified by R.
> **IF:** more than one entity relation satisfies the above conditions for the set of attributes, X;
> **THEN:** the user must determine the exact participating entity type.
> **JUSTIFICATION:** Similar to the determination of owner entity types for the weak entity types case, conditions 1 and 2 above are necessary conditions for the entity type identified by A to be a participating entity type of the relationship type identified by R. If A is not

unique in satisfying those conditions, then further information which can only be supplied by the user is needed.

Consider WORK-FOR in Table 4. Suppose that the following two inclusion dependencies hold:

```
WORK-FOR.[SSN]    ≪EMPLOYEE.[SSN]
WORK-FOR.[DEPTNO]≪DEPARTMENT.[DEPTNO].
```

The extraction process identifies a binary relationship type between *Department* and *Employee* as shown in Fig. 10.

Specific relationship relation For each specific relationship relation, the extraction process first identifies entity types for general key attributes, and then identifies an n-ary relationship type among all of the entity types identified by the primary key and general key attributes. The identification rule is:

IF: there is a specific relationship relation, R with general key attributes, Y;
THEN:
(a) identify a strong entity type for the general key attribute, Y, and
IF: R contains multiple general key attributes;
THEN: the user must determine how many new entity types need to be identified.
(b) identify an n-ary relationship type among participating entity types.
JUSTIFICATION: General key attributes indicate the existence of additional entity types which must be identified first. Such entity types can be represented in the EER model by containing general key attributes as their only attributes. However, if R contains multiple general key attributes, then the general key attributes may represent more than one entity type.

The participating entity types of this relationship are determined by the rule which is the same as that used for the regular relationship relations, except that the new entity types are identified by the general key attributes. Consider SHIPMENT and ORDER in Table 4 and an inclusion dependency:

```
SHIPMENT.[ORDID]≪ORDER.[ORDID].
```

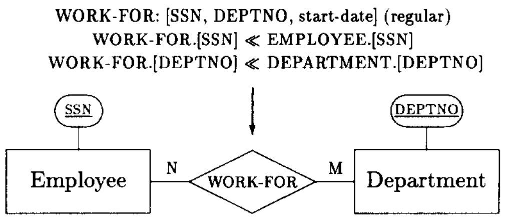

WORK-FOR: [SSN, DEPTNO, start-date] (regular)
WORK-FOR.[SSN] ≪ EMPLOYEE.[SSN]
WORK-FOR.[DEPTNO] ≪ DEPARTMENT.[DEPTNO]
SSN
DEPTNO
Employee
N
WORK-FOR
M
Department

Fig. 10. A regular relationship relation identifies an n-ary relationship.

For a specific relationship relation such as SHIPMENT, one or more new entity types must be identified first. For example, PACK#, the general key attribute, identifies an entity type which contains it as the only attribute. The user must provide a name for this new entity type. Then, a relationship type is identified among the participating entity types whose keys form the primary key of the specific relationship relation. For SHIPMENT, a binary relationship type is identified between the entity type identified by ORDER and the entity type identified by PACK#, which is called *Package*. Figure 11 shows the translation of a specific relationship relation and its representation in the EER diagram.

5.4.5 Aggregate relationships

A foreign key in a relationship relation indicates that an aggregate relationship exists between an entity type and a relationship type. The identification rule is:

IF:

(1) a foreign key, x, of a relationship relation, R, appears as a key, X, of an entity relation (strong or weak), A, and

(2) there is an inclusion dependency between them, $R.x \ll A.X$;

THEN:

(a) employ the aggregation abstraction to group the n-ary relationship type identified by R and its participating entity types as a composite entity type [21],

(b) provide a name for this composite entity type, and

(c) identify an aggregate relationship.

There are two participating entity types for an aggregate relationship; one is the composite entity type identified by the relationship relation containing the foreign key and the other is the entity type identified by the relation whose key is the same as the foreign key.

IF: more than one entity relation, A, satisfies the above conditions;

THEN: the user must determine the exact participating entity type.

JUSTIFICATION: From Table 1, the case where there is a relationship between an entity type with (0,1) or (1,1) *Min/Max* values and a relationship type that is represented by a foreign key. This is converted into an aggregate relationship. Similar to the determination of participating entity types for the foreign key of entity relation case, conditions 1 and 2 above are necessary conditions for the entity type identified by A to be a participating entity type of the aggregate relationship type identified by the foreign key, x. If A is not

SHIPMENT: [PACK#, ORDID, ship-date, carrier-id] (specific)
ORDER: [ORDID, order-date, prodid, custid] (strong)
SHIPMENT.[ORDID] $\ll$ ORDER.[ORDID]

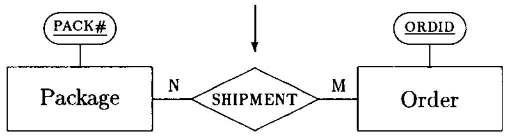


Fig. 11. A specific relationship relation identifies an n-ary relationship.

unique in satisfying those conditions, then further information which can only be supplied by the user is needed.

Consider CARRIER and SHIPMENT in Table 4 and an inclusion dependency:

SHIPMENT.[carrier-id] $\ll$ CARRIER.[CARRIER-ID].

In order to identify the relationship represented by the foreign key, 'carrier-id', the *n*-ary relationship identified by SHIPMENT must first be aggregated as a composite entity type. This composite entity type can have a name other than the relationship's name. Then, an aggregate relationship is identified between the composite entity type and the entity type containing 'carrier-id' as its key. The cardinality ratio for an aggregate relationship could be 1:1 or 1:N. The default is 1:N. Figure 12 illustrates the identification and representation of an aggregate relationship in the EER diagram.

5.5. Assignment of attributes

The primary key attributes for each relation are used either as the key of the corresponding structure in the EER model, or to identify the relationship type. The foreign key attributes are used to identify binary relationships, so that they are not assigned to any modelling structure in the EER model, except when they are also candidate keys. The candidate keys for each strong entity relation are assigned as descriptive attributes to the corresponding entity type.

The extraction process, therefore, only needs to consider how to assign remaining non key attributes as descriptive attributes to entity and relationship types. The manner in which non key attributes are assigned depends on:
- the relation's type (an entity relation or a relationship relation);
- whether the relation contains foreign keys; and
- the meaning of the particular non key attribute in the relation.

Since the meaning of non key attributes normally cannot be obtained from the database

CARRIER: [CARRIER-ID, name, address] (strong)
SHIPMENT: [PACK#, ORDID, ship-date, carrier-id] (specific)
SHIPMENT.[carrier-id] $\ll$ CARRIER.[CARRIER-ID]

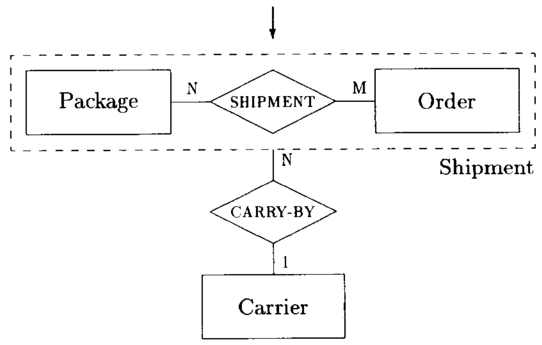


Fig. 12. Identification of an aggregate relationship.

system, the user should provide such information when necessary. The assignment is carried out based upon the following rules.

(1) **IF:** an entity relation does not contain any foreign key;
THEN: assign all of the non key attributes of this relation to be the descriptive attributes of the corresponding entity type.
JUSTIFICATION: An entity relation without any foreign key identifies only an entity type; therefore, all the non key attributes must belong to the entity type identified by this relation.
Consider the following relation:

```
PERSON:[SSN, name, address] (strong).
```

The non key attributes, 'name' and 'address', are assigned as descriptive attributes of the entity type *Person*.

(2) **IF:** an entity relation has foreign key attributes;
THEN: non key attributes can be assigned to be descriptive attributes of either the entity type identified by this relation or one of the binary relationship types identified by the foreign keys of this relation.
JUSTIFICATION: An entity relation with a foreign key identifies not only an entity type, but also a binary relationship type. Therefore, each non key attribute except for candidate key attributes can belong to either the entity type identified by this relation, or the binary relationship type identified by the foreign key of this relation.
Consider the following relation:

```
MANAGER: [SSN, rank, promotion-date, deptno] (strong).
```

The non key attribute, 'rank', should be assigned as a descriptive attribute of the entity type *Manager*. However, the non key attribute, 'promotion-date', can be considered as a relationship attribute for the relationship type identified by the foreign key, 'deptno'.

(3) **IF:** a relationship relation does not have any foreign key;
THEN: assign all of the non key attributes as the descriptive attributes of the n-ary relationship type identified by this relation.
JUSTIFICATION: A relationship relation without any foreign key identifies only a relationship type, so that all of the non key attributes must belong to the relationship type identified by this relation.
Consider the following relationship relation:

```
WORK-FOR: [SSN, DEPTNO, start-date] (regular).
```

The non key attribute of this relation, 'start-date', is assigned as the relationship's attribute for the relationship type, *work-for*.

(4) **IF:** a relationship relation has a foreign key;
THEN: each non key attribute is assigned to be a descriptive attribute of either:

(a) the *n*-ary relationship type identified by this relation, or
(b) one of the binary relationship types identified by the foreign keys of this relation.
JUSTIFICATION: A relationship relation with a foreign key identifies not only a relationship type, but also an aggregate relationship. Therefore, each non key attribute can belong to either the relationship type identified by this relation, or the aggregate relationship identified by the foreign key of this relation.
Consider the following relationship with a foreign key:

```
SHIPMENT: [PACK#, ORDID, ship-date, carrier-id] (specific).
```

The non key attribute, 'ship-date', can belong to either one of two relationship types. One is identified by the relation, SHIPMENT; the other one is identified by the foreign key, 'carrier-id'.

5.6. Verification and validation of the extraction process

There are several issues to be addressed with regard to evaluating reverse engineering methodologies. The first question is one of correctness: how faithfully will a database be represented by the EER model produced from it? This is a measure of the quality of the 'Reverse Engineering' arrow in Fig. 1. However, what is a faithful representation in this context? Since the spirit of reverse engineering in general is to produce a set of specifications which would lead to the implementation in question, a faithful representation here corresponds to an EER model which good database design principles would map into the database in question. We have addressed this issue in the development of the methodology itself, by characterizing the EER-to-database translation as a set of functions, and then inverting these functions to give reasonable interpretations of given database structures in EER model terms. This is summarized by Tables 1 and 3, and the justifications for the rules given in Section 5.

A second issue is whether all relational structures that would be found in database implementations are accounted for in the translation rules (completeness), and if not, how the rules which the methodology presently incorporates would translate them (robustness). This is largely an empirical question which can best be answered by testing on real databases. To this end, a prototype system based on the methodology, called the Knowledge Extraction System (KES), has been developed. KES is written in Arity Prolog and Microsoft C, integrated with the Oracle RDBMS, and runs on an IBM PC. See Chiang [9] for details. Testing has been done on a small sample of databases so far, for example the schema shown in Fig. 2 and the resulting EER model which is shown as a graphical diagram in Fig. 13. To simplify the presentation of this diagram, the *is-a* relationship between *Customer* and *Person*, and the generalization hierarchy for the *generic* entity type, *Person* are not shown in the figure. More testing is planned for the future, and we expect new discoveries will lead to enhancement of the current methodology.

The third issue is whether the resulting EER model is a good representation of the real-world domain which the database is supposed to model; this corresponds to the 'Validation' arrow of Fig. 1. This is a multi-faceted problem.

First, if the design and maintenance decisions leading to the current state of the database

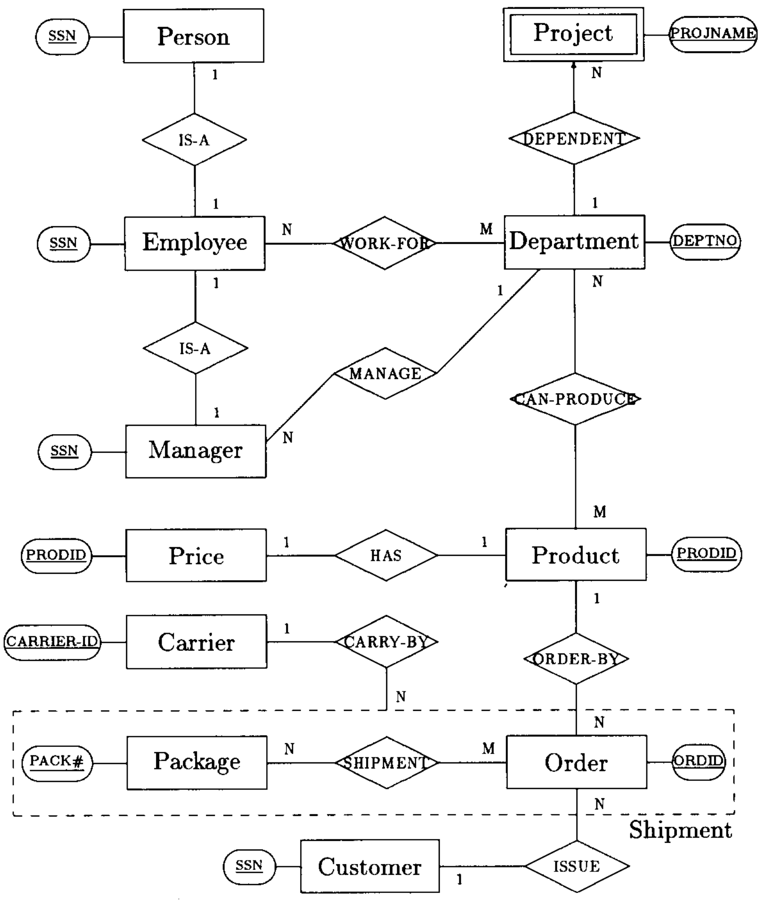


Fig. 13. The EER model obtained by the extraction process.

design were inappropriate for the faithful representation of the domain, then it is unlikely that any automated reverse engineering will lead to a fully satisfactory model by itself. Thus the EER model will have to be evaluated by someone familiar with both the domain in question and what the database is intended to model about that domain, and improved as needed. It is, however, likely to be easier to perform this 'editing' task than it would be to begin the design from scratch, and it is certainly easier to do so by starting from an EER model than from the database itself.

Second, if good non-EER design methods are followed in the original design and maintenance, will the EER model resulting from applying the methodology be a faithful rendering of the original design intentions? Note that this is not primarily a reverse engineering issue, but rather, an issue of the comparative expressive powers and robustness of alternative design methods. This is largely an empirical question which is probably best answered by controlled experiment. This would involve creating databases using alternative design methods, and then applying KES to the resulting databases and evaluating the EER models it produces. Such testing would be likely to lead to a more robust system, and such experiments are being contemplated for future research.

6. Summary and conclusions

A methodology for performing reverse engineering of relational databases has been presented. This methodology extracts an EER model from an existing database by analyzing, not only the data scheme, but also data instances. The issues with regard to the validation and verification of the extraction process are also discussed. A prototype system, called the Knowledge Extraction System (KES), has been developed for these purposes.

There are a number of possible extensions to this research. First, we can consider relaxing the assumptions about input databases. For example, when the input database contains erroneous data in the key attributes, certain inclusion dependencies should not be rejected. Second, the reverse engineering process could consider not only relations, but also views of the input databases. Third, the EER model obtained from the extraction process could be refined and enhanced by incorporating additional heuristics and domain knowledge. For example, redundant attributes for entity types in *is-a* hierarchies could be detected and removed according to the property inheritance principle [6].

Fourth, a set of integrity constraints can be proposed based on the resulting EER model and discovered inclusion dependencies. Finally, the resulting EER model could be used to generate a new data schema for the existing database. For example, the representation for some relationships may be changed based on the cardinality values discovered from data instances; this is the *reengineering* process of databases [10].

References

[1] C.W. Bachman, A personal chronicle: Creating better information systems, with some guiding principles, *IEEE Trans. Knowledge and Data Engineering* 1(1) (Mar. 1989) 17–32.

[2] C. Batini, M. Lenzerini and S.B. Navathe, A comparative analysis of methodologies for database schema integration, *ACM Comput. Surveys* 18(2) (Dec. 1986) 323–364.

[3] C. Batini, S. Ceri and S.B. Navathe, *Conceptual database design: An Entity-Relationship Approach*, (Benjamin/Cummings, Mento Park, CA, 1992).

[4] P.A. Bernstein, Synthesizing third normal form relations from functional dependencies, *ACM Trans. Database Syst.* 1(4) (Dec. 1976) 272–298.

[5] D. Bitton, J. Millman and S. Torgersen, A feasibility and performance study of dependency inference, *Proc. Fifth Internat. Conf. on Data Engineering* (Feb. 1989) 635–641.

[6] R.J. Brachman, What IS-A is and isn't: An analysis of Taxonomic links in semantic networks, *IEEE Comput.* 16(10) (Oct. 1983) 30–36.

[7] M.A. Casanova, R. Fagin and C. Papadimitriou, Inclusion dependencies and their interaction with functional dependencies, *J. Comput. System Sci.* 28(1) (Feb. 1984) 29–59.

[8] P.P. Chen, The Entity Relationship Model – Toward a unified view of data, *ACM Trans. Database Syst.* 1(1) (Mar. 1976) 9–36.

[9] R.H.L. Chiang, A knowledge-based system for performing reverse engineering of relational databases, *Decision Support Syst.*, forthcoming March, 1995.

[10] E.J. Chikofsky and J.H. Cross II, Reverse engineering and design recovery: A taxonomy, *IEEE Software* (Jan. 1990) 13–17.
[11] Computer-aided software engineering: Reverse-engineering tools, *Computerworld* (April 9, 1990) 78–80.
[12] C.J. Date, *An Introduction to Database Systems: I & II*, 4th ed. (Addison-Wesley, Reading, MA, 1986).
[13] K.H. Davis and A.K. Arora, Converting a relational database model into an Entity-Relationship model, in S.T. March, ed., *Entity-Relationship Approach* (Elsevier Science, Amsterdam, 1988) 271–285.
[14] S.R. Dumpala and S.K. Arora, Schema translation using the Entity-Relationship approach, in P.P. Chen, ed., *Entity-Relationship Approach to Information Modelling and Analysis* (ER Institute, 1981) 339–360.
[15] R. Elmasri, J. Weeldreyer and A. Hevner, The Category Concept: An extension to the Entity-Relationship Model, *Internat. J. Data and Knowledge Engrg.* 1(1) (May 1985) 75–116.
[16] R. Elmasri and S.B. Navathe, *Fundamentals of Database Systems* (Benjamin/Cummings, Menlo Park, CA, 1989).
[17] J.-L. Hainaut, Database reverse engineering: Models, techniques and strategies, in: *Proc. Tenth Internat. Conf. on Entity-Relationship Approach* (1991) 729–741.
[18] Wenguang Ji et al., An algorithm converting relational schemas to nested Entity Relationship schemas, in *Proc. Tenth Internat. Conf. on Entity-Relationship Approach* (1991) 231–246.
[19] P. Johannesson and K. Kalman, A method for translating relational schema into conceptual schemas, in *Proc. Eighth Internat. Conf. on Entity-Relationship Approach* (1989) 279–293.
[20] K. Kalman, Implementation and critique of an algorithm which maps a Relational Database to a Conceptual Model, *SYSLAB Working Paper 151* (June 1989).
[21] H.F. Korth and A. Silberschatz, *Database System Concepts* (McGraw-Hill, New York, 1986).
[22] V.M. Markowitz and J.A. Makowsky, Identifying Extended Entity-Relationship object structures in relational schemas, *IEEE Trans. Software Engrg.* 16(8) (Aug. 1990) 777–790.
[23] T. Martyn and T. Hartley, Semantics & logical errors using SQL, in *Database programming & design* (June 1990) 51–56.
[24] S.B. Navathe and A.M. Awong, Abstracting relational and hierarchical data with a semantic data model, in S.T. March, ed., *Entity-Relationship Approach* (Elsevier Science, Amsterdam, 1988) 305–333.
[25] S.B. Navathe and R. Elmsari, Integrating user views in database design, *IEEE Comput.* 19(1) (Jan. 1986) 50–62.
[26] H.A. Schmid and J.R. Swenson, On the semantics of the Relational Data Model, *Proc. 1975 ACM SIGMOD Internat. Conf. on the Management of Data* (June 1975) 211–223.
[27] J. Smith and D. Smith, Database abstractions: Aggregation and generalization, *ACM Trans. Database Syst.* 2(2) (June 1977) 105–133.
[28] V.C. Storey, Relational database design based on the Entity-Relationship model, *Data & Knowledge Engrg.* 7(1) (Nov. 1991) 47–83.
[29] T.J. Teorey, Y. Dongqing and J.P. Fry, A logical design methodology for relational databases using the Extended Entity Relationship Model, *ACM Comput. Surveys* 18(2) (June 1986) 197–222.
[30] D. Tsichritzis and E. Lochovsky, *Data Models* (Prentice-Hall, Englewood Cliffs, NJ, 1982).
[31] R.B. Wilmot, Foreign keys decrease adaptability of database designs, *Comm. ACM* 27(12) (Dec. 1984) 1237–1243.
[32] J. Winans and K.H. Davis, Software reverse engineering from a currently existing IMS database to an Entity-Relationship Model, in H. Kangassalo, ed., *Entity-Relationship Approach: The Core of Conceptual Modelling* (Elsevier Science, Amsterdam, 1991) 333–348.

Roger H.L. Chiang, Lecturer, School of Accountancy and Business at Nanyang Technological University, Singapore, has a B.S. degree in Management Science from National Chiao Tung University, Taiwan, M.S. degrees in Computer Science from Michigan State University and in Business Administration from University of Rochester, and a Ph.D. degree in Computers and Information Systems from University of Rochester. His Ph.D. dissertation focuses on developing a methodology for reverse engineering of relational databases. He is a runner-up of the 1993 Doctoral Dissertation Competition of the International Conference on Information Systems (ICIS). His research interests include database reverse engineering, data management, knowledge-based systems, and conceptual data modelling. He is a member of AAAI, ACM, IEEE, and TIMS.

Terry Barron is Assistant Professor of Computers and Information Systems at the William E. Simon Graduate School of Business Administration, University of Rochester, Rochester, NY. He received his Ph.D. in 1987 from the Graduate School of Business Administration, University of Washington (Seattle). His research interests include economics of computer system management, and information system analysis and design.

Veda C. Storey, Assistant Professor of Computers and Information Systems at the William E. Simon Graduate School of Business Administration, University of Rochester, has research interests in database management systems and artificial intelligence. Her research has been published in *ACM Transactions on Database Systems*, *Information Systems Research*, *Data and Knowledge Engineering*, and the *Very Large Data Base Journal*. She is the author of *View Creation: An Expert System for Database Design*, a book based on her doctoral dissertation, and published by ICIT (International Center for Information Technology) Press in 1988.

Dr. Storey received her doctorate in Management Information Systems from the University of British Columbia, Canada, in 1986. She earned a Master of Business Administration degree from Queen's University, Ontario, Canada, in 1980, and a Bachelor of Science degree (with distinction) from Mt. Allison University, New Brunswick, Canada in 1978. In addition, she received in 1978 her Associate of the Royal Conservatory of Music for flute performance from the University of Toronto.