Implementation and Critique of an algorithm which maps a Relational Database to a Conceptual Model

Katalin Kalman

Department of Computer and Systems Sciences
Stockholm University and the Royal Institute of Technology
Stockholm, Sweden

ABSTRACT

The degree to which the creation of a conceptual model can be mechanized is a topic of investigation of considerable interest within the research area of semantic data modeling. This paper examines one such example in which the feasibility of mechanically creating a conceptual schema from a relational database is explored. The approach chosen to carry out the task of this translation is an algorithm

1 INTRODUCTION

Organizations tend to possess a large number of databases which are organized with the objective of making the data contained therein available to various applications. Many of these databases have evolved over a long period of time and the applications for which they were originally intended are no longer used. Instead, the databases are needed for new applications. Unfortunately vital semantic information often gets lost in the process of encoding information to create a database. This often results in a situation where the ones with access to the original information are unavailable and those inheriting these databases lack the understanding for using and maintaining them. The main purpose of *reverse modeling* is to recapture this missing information in currently existing databases.

A method of reverse modeling, which maps a relational database to a conceptual schema is presented, examined and analyzed here. It is a method proposed by Navathe and Awong in [4]. In [4], two algorithms are presented. One of these is a method, based on similarity of attribute names in the relations of the database. Attributes which are the same are expected to have identical names throughout the database. It is by identifying like attributes of different

relations, that connections between the relations of the database are established.

The data initially available to the mapping process consists of the relations of a database. The relations are always in the third normal form and are given without instances.

2 THE EXTENDED ENTITY RELATIONSHIP MODEL

The algorithm which is to be examined here is a ten step procedure which converts a relational database to an ECR (Entity-Category-Relationship) model, a case of an EER (Extended Entity-Relationship) model..

An ECR model is a semantic model which is composed of entities, relationships, subclasses, and generalization hierarchies. An entity is an object in a particular environment. Entities are connected to each other by means of relationships. A subclass is an entity which fulfills a subclass role in a subclass/superclass relationship between two entity types. That is to say, the entity which is a subclass is a subset of the entity which is its superclass. A generalization category (hierarchy) over a number of other entities is an entity which fulfills a superclass role to a number of other entities which are its subclasses.

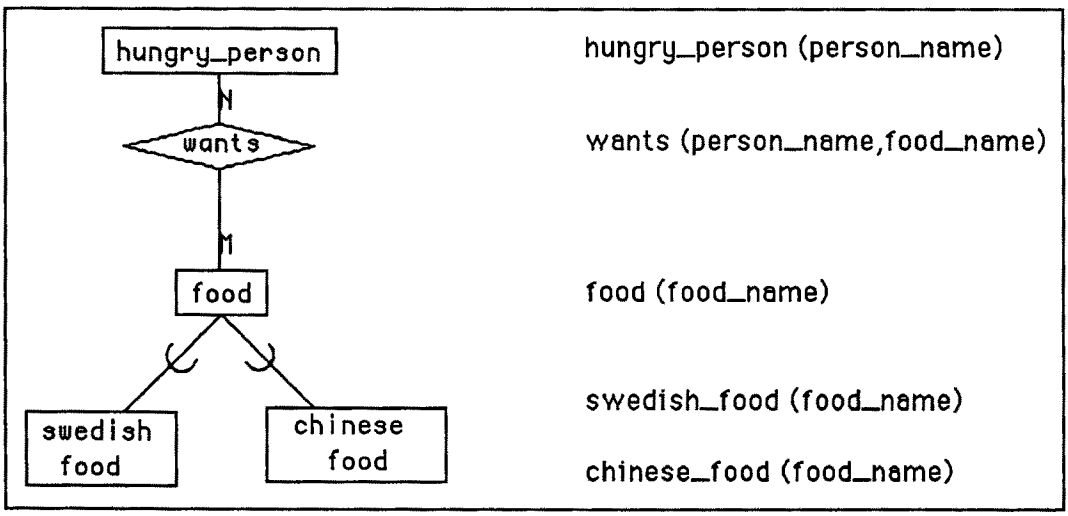


figure 2.1 The EER model

To illustrate the components of the EER model used in this paper, a graphical representation of a conceptual model as well as the corresponding database relations are presented in the example above (see fig. 2.1). In this representation, entities are denoted by rectangles, relationships by diamonds, and subclasses by concave arcs placed on the connecting line between the subclass entity and its superclass. In this example *wants* is a relationship between entity types *hungry_person* and *food*, the entities *swedish_food* and *chinese_food* are subclasses to the entity type *food*.

3 A TRANSFORMATION STRATEGY IN NINE STEPS

The algorithm, which maps a relational database to an ER schema, has been implemented, with minor alterations and a few omissions, in MAKEMODEL, a program written in Prolog. Input to the program is a relational schema, that is, relation and attribute names without instances represented as a set of Prolog facts. These facts are then processed by a nine step procedure which creates an extended Entity-Relationship semantic data model containing subclasses as well as mappings of functional dependencies. Generalization categories have been omitted here as the usage of the concept is restricted to Navathe and Awong's paper and has, thus, been deemed too narrow to be considered of enough value to be employed in this implementation.

The aim of this method is to atomize the process of transformation as much as possible. However, since more information is contained in the conceptual schema than in the original database, the process cannot be totally atomized. Whenever ambiguities, that cannot be resolved mechanically, arise, MAKEMODEL makes use of user interaction to obtain the clarification it needs to be able to proceed.

The strategy used by MAKEMODEL to transform Prolog facts depicting relations in a database to prolog facts representing a conceptual model can, thus, be described as a process which methodically fine tunes the database, using a mixture of automatic modification and user interaction. Although it is a nine step algorithm, it can also be divided into two distinct parts. Step1 constitutes the first part, the second part consists of the remaining eight steps. Step1 uses the candidate keys to restructure the relations for the purpose of making it possible, at a later stage, to find relationships between them. The candidate keys are in effect neutralized here as they no longer have any value to the transformation process after this step. All relations as well as their attributes are classified in this first step.

A detailed account of the nine step algorithm used by MAKEMODEL is given below:

STEP1 - Three actions that perform different types of candidate key substitutions are carried out here:

action1- The primary key of each relation in the relational database is read. After reading a primary key, the candidate keys of the rest of the relations are scanned. If a candidate key is found which is identical to the primary key under examination, and there exists a subclass-superclass relationship between the two relations, that candidate key is changed to be the new primary key of the relation and the relation's former primary key is changed to be a new candidate key. To find out if such a subclass superclass relationship exists.

An example of how MAKEMODEL processes such a case:

Input: listing produced by MAKEMODEL:

account_holder
Primary Key: [acctnum]

```
person
Primary Key: [person_id]
Candidate Key: [acctnum]

company
Primary Key: [company_id]
Candidate Key: [acctnum]
```

Interaction between MAKEMODEL and the user:

```
In order to establish whether there exists a sub/super class
relationship between the relations shown below,
please answer the following question(s) by typing yes or no.
Is there a sub/super class relationship between:
company and account_holder?
y
Is there a sub/super class relationship between:
person and account_holder?
y
```

Result of processing after action1:

```
account_holder
Primary Key: [acctnum]

person
Primary Key: [acctnum]
Candidate Key: [person_id]

company
Primary Key: [acctnum]
Candidate Key: [company_id]
```

The purpose of candidate key substitutions performed by action1 is to discover some of the sub/superclass relationships between relations and to prepare these for further questioning later. In step9 all sub-superclass relationships are processed.

action2- The primary key of each relation in the relational database is read. After reading a primary key, the candidate keys of the rest of the relations are scanned. If the primary key is found to contain a candidate key of another relation, the part of the primary key which is equivalent to one of the candidate keys is replaced by the primary key of the relation which contains that candidate key.

An example of how MAKEMODEL processes such a case:

Input: listing produced by MAKEMODEL:

```
car_sale
```

Primary Key: [buyer_id,eng_serial_num,seller_id]
None Key Attributes: [sale_date]

car
Primary Key: [license_num]
Candidate Key: [eng_serial_num]
None Key Attributes: [make,model]

Result of processing after action2:
car_sale
Primary Key: [buyer_id,license_num,seller_id]
None Key Attributes: [sale_date]

car
Primary Key: [license_num]
Candidate Key: [eng_serial_num]
None Key Attributes: [make,model]

The purpose of candidate key substitutions performed by action2 is to provide relations which have nothing linking them to each other by common elements in their primary keys with such links. The relation whose primary key was changed, now has a potential to become a relationship between two or more entities at some future time.

action3- The candidate key of each relation in the relational DB is read. After reading a candidate key, the primary keys of the rest of the relations are scanned. If the candidate key of the relation under examination is formed by concatenating two or more primary keys, that candidate key becomes the new primary key of the relation and the previous primary key is made into a new candidate key.

Input: listing produced by MAKEMODEL:
course_offering
Primary Key: [section_num]
Candidate Key: [course_num,instructor_num]

instructors
Primary Key: [instructor_num]

courses
Primary Key: [course_num]

Result of processing after action3:
course_offering
Primary Key: [course_num,instructor_num]
Candidate Key: [section_num]

The purpose of candidate key substitutions performed by action3 is to identify relations which will later become relationships between entities.

The next stage is that of classification. First, the relations are classified into primary type1 (PR1), primary type2 (PR2), secondary type1 (SR2), and secondary type2 (SR2):

PR1 - Relations whose primary key does not contain a key of another relation. (Will map to entities).

PR2 - Relations whose primary key is ID dependent on the key of a PR1 relation. To decide whether an ID dependence exists between two relations, first the primary key of a relation is examined to see if it contains the primary key of a PR1 relation. If this proves to be the case, the user is asked to confirm or deny an ID dependence between the two relations. (Will map to weak entities).

SR1 - Relations whose primary key is fully formed by concatenating the primary keys of PR1 and/or PR2 relations. (Will map to relationships).

SR2 - Relations whose primary key is partially formed by concatenating the primary keys of PR1 and/or PR2 relations. (Will map to k-ary relationships).

Next, the attributes of all the relations are classified into four categories: Key Attribute Primary (KAP), Key Attribute General (KAG), Foreign Key Attribute (FKA), and None Key Attribute (NKA).

KAP - Attributes in the primary key of a secondary relation which also constitute a primary key of a primary relation. These are used later (in step 6) when processing SR2 relations. These relations map to relationships, which connect entities whose entity identifiers are the same as the KAP of the SR2 relation to new entities created from KAGs.

KAG - The remaining attributes in the primary key of a secondary relation which are not KAPs. These give rise to new entities used in the transformation of SR2 relations.

FKA - None primary key attribute that is also a primary key of another relation. Used (in step 7) to establish a relationship between the entity created by the relation which contains this attribute as an FKA and the relation which contains the attribute as its primary key.

NKA - None primary key attributes which are not FKAs.

STEP2 - When two or more primary relations have identical primary keys and there exists a secondary relation which contains this key, it must be determined which of these primary relations is related to the secondary relation. This is resolved by asking the user to identify the primary relation which is relevant.

Interaction between MAKEMODEL and the user:

```
The attribute ss_num in flies comes either from emp or pilot.
Does ss_num come from emp?
n
Does ss_num come from pilot?
y
```

Those primary relations identified as not being relevant to the secondary relation in question are gathered together in a list and saved to be used later (in steps 5 and 6).

STEP3 - Entity types are created here. For each PR1 relation, an entity type is asserted to the prolog database.

STEP4 - Weak entity types are created. For each PR2 relation, a weak entity type is asserted to the prolog database.

STEP5 - Relationship types are created. First all SR1 relations are read. Then those primary relations whose primary keys form the primary key of the SR1 relation are gathered together in a list. This list is then updated by deleting those relations (now entities) which have been marked in step2 as being irrelevant to the SR1. A relationship type containing the list of the entities which are connected to each other through this new relationship. is created and asserted to the prolog database.

STEP6 - Relationship types are created from SR2 relations that contain both KAP and KAG type attributes. For each SR2 relation a list is produced containing the names of the entities whose identifiers are the same as the KAPs of the SR2 (the primary relations whose primary key is the same as the KAP) as well as the names of the KAG attributes of the SR2. Relations marked as irrelevant for the SR2 are removed as in step5.

STEP7 - Relationships derived from FKA type attributes are created. For each relation that contains an FKA type attribute, a relationship between the entity formed by that relation and the entities whose identifiers are the same as the FKA attributes contained in the relation is defined.

STEP8 - Binary relationships are labeled with their corresponding functional dependencies. Relationships, mapped from SR1 and SR2 relations, that connect exactly two entities are processed separately from other relationships. The user is asked to state the functional dependency which prevails between the two entities connected by a binary relationship.

Interaction between MAKEMODEL and the user:

In order to assess the functional dependency between |courses| and |instructors| in relationship <course_offering>, please chose the correct dependency listed below by typing the number 1, 2, 3, or 4.

1. There is a functional dependency from |courses| to |instructors|.
2. There is a functional dependency from |instructors| to |courses|.
3. Both of the above functional dependencies exist.
4. None of the above functional dependencies exist.
1

STEP9 - Subclasses are created. When entities with identical identifiers are detected, the user is asked to state whether a sub-superclass relationship exists between these entities.

Interaction between MAKEMODEL and the user:

Please indicate whether there exists a sub/super class relationship between the following entities.

E1->E2 denotes that E1 is a subclass of E2.

```
emp->person
y
emp->pilot
n
pilot->emp
n
person->pilot
n
pilot->person
y
```

For each subclass relation identified, **subclass(E1,E2)** is asserted. **E1** represents an entity which is a subclass of the entity represented by **E2**.

4 MAKEMODEL: AN IMPLEMENTATION OF THE ALGORITHM

Implementing a method renders it testable. This facilitates the process of evaluation, which is done for the purpose of determining the validity of a method. It is, first, by testing with different examples that it becomes possible to infer something about the accuracy of a method.

MAKEMODEL, an implementation of the algorithm presented in [4], was constructed for the purpose of examining that method. A program description is presented below.

Program Description:

To activate MAKEMODEL, the user types the word start followed by a period. MAKEMODEL, then, begins its processing by requesting the user to select a relational database. The user may, choose to use an existing database, create his/her own database, or supplement an existing database with new relations.

After establishing an input database to MAKEMODEL, the user is given an opportunity to view the database. If this option is chosen, then every relation in the database gets displayed. For each relation, its name is printed out followed by the attributes which constitute its primary key, the attributes which constitute any and all candidate keys the relation may embody, as well as any existing none key attributes.

The nine steps making up the algorithm are executed next. This is done by 'process RDB to CM' which is the principal component of MAKEMODEL as it is where the mapping from a relational database to a conceptual model takes place. All questions pertinent to the processing of the particular database, given as input to the current run, are asked here and the generated structures corresponding to the conceptual model are asserted to the Prolog data base.

Finally, the user is given a chance to view the conceptual model created by MAKEMODEL. This step may be bypassed, in which case, a concluding remark is printed out and MAKEMODEL is exited. If the user wishes to see the model, then information about each entity and relationship produced is printed out and graphical representations of relationships between the entities are drawn out on the screen.

5 CRITIQUE OF THE ALGORITHM

Prior to reaching the level of competence necessary to criticize any method, it is important to gain a reasonably good understanding of the method in question. One way to accomplish this is by using pragmatic means. The behavior of the method could be observed under various conditions. This would make it possible for different aspects of the method to seen, thereby enabling one to acquire a better understanding of the method as a whole.

In this case, the algorithm has been closely scrutinized by testing MAKEMODEL quite thoroughly. A vast number of examples were used as input to MAKEMODEL and the results carefully noted and recorded.

To appreciate the results obtained from using test examples, it can be helpful to keep in mind the different types of relationships which can arise and how MAKEMODEL creates these. A brief recapitulation follows.

Four different types of relationships can prevail between entities. The first is a relationship between a subclass entity and its superclass. The second is a relationship between an entity and a weak entity. The third is a relationship between two or more entities in which the relationship has a name, a relationship identifier and none key attributes. Finally, there is the relationship between two entities where the relationship has neither a name nor identifier.

The possibility of a relationship between a subclass entity and its superclass is recognized whenever there exists two or more relations with the same primary key. This situation is first isolated in step1.action1, through a user question, whenever a candidate key of a relation matches the primary key of another relation. The other two candidate key substitutions can also give rise to this situation. The place where sub/superclass relationships are actually created is step9. Weak entities are recognized via user interaction in step1. They become classified as PR2 (primary relations) and the relationship is finally created in step4. Ordinary relationships between two or more entities are recognized by the fact that their primary keys contain primary keys of other relations. These relations get labeled as SR1 or SR2 (secondary relations) in step1 and are processed in different ways, depending on various conditions, in steps 5,6 and 8. The last type of relationship is the FKA. These are created from relations containing attributes which are also primary keys of other relations. These attributes are labeled as FKA in step1 and are then set up as relationships in step7.

It is not difficult to see that the creation of relationships is a rather elaborate process. The manner in which a relation is turned into an entity or a relationship is rather opaque and it is, thus, not intuitively clear that the model generated is the correct one.

The examples used to test the algorithm, were, for the most part, arbitrary databases selected from various domains. Many were also somewhat large and rather intricate. The reason for choosing large examples, was that the shortcomings of the algorithm are, more likely, found by using examples where more complex situations are depicted.

A listing of ten important and basic problems inherent in the algorithm and made apparent by testing MAKEMODEL is presented below. Each problem is described, the correct model which should have resulted is shown, and a solution is proposed.

(1) - Multi level ID dependence

Problem description:

The situation in which an entity is ID dependent on another entity which in turn is ID dependent on a third entity cannot be produced. The reason for this is that according to the algorithm an entity can only be ID dependent on a PR1 relation.

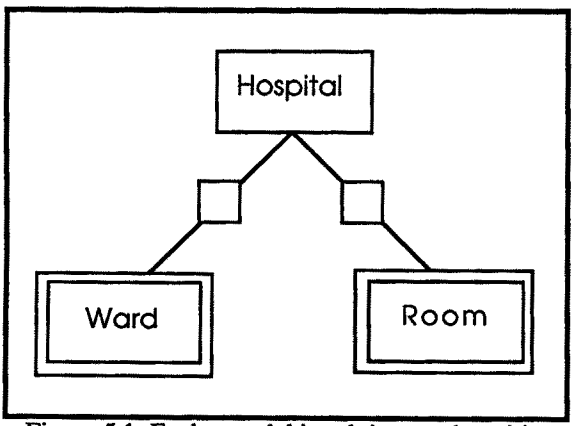


Figure 5.1 Faulty model involving weak entities

In this example the entity Room is ID dependent on the entity Ward which in turn is ID dependent on the entity Hospital, but this cannot be represented by MAKEMODEL. Instead, the two relations Ward and Room become weak entities ID dependent on Hospital, an entity created out of a PR1 relation.

The resulting faulty model is shown in Figure 5.1 to the left.

Proposed solution:

This can be solved by letting a relation be ID dependent on none PR1 relations as well as on PR1 relations.

Figure 5.2 below shows the correct conceptual model for the example discussed above.

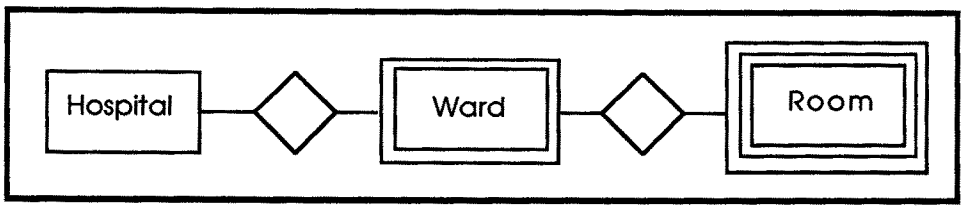


Figure 5.2 Correct model involving weak entities

(2) - Sub/super class relationships and the creation of indirect links between these.

Problem description:

When two or more levels of sub/super class relationships exist, the indirect links should only be represented implicitly in the conceptual model. i.e. if there were a subclass relationship from an entity C to an entity B and a subclass relationship from entity B to an entity A, then the subclass relationship from C to A should not be depicted in the conceptual model. These indirect links do, however, become allocated. Since sub/super class relationships are established through user interaction, the question as to the existence of a connection between A and C would be answered in the affirmative and a direct link between A and C would then be set up in the conceptual model.

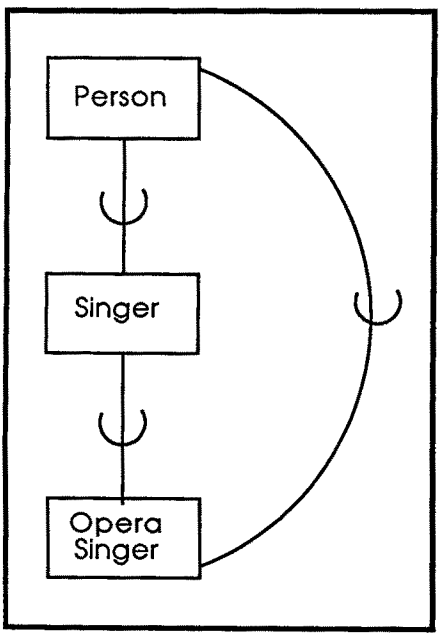


Figure 5.3 Indirect link

The link between Opera singer and Person should not be represented explicitly in the conceptual model.

Figure 5.3 to the left shows the faulty model created by MAKEMODEL.

Figure 5.4 to the right shows the correct conceptual schema.

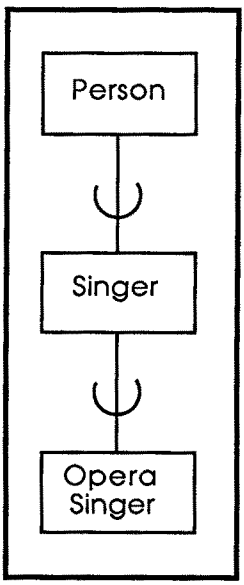


Figure 5.4

Indirect link not represented

Proposed solution:

This problem can be solved by deleting indirect links after all super/subclass links for a particular primary key have been specified by the user.

(3) - FKA relationship between an entity and one participating in a sub/superclass relationship.

Problem description:

Since a subclass superclass relationship between entities is mapped from relations which have identical primary keys, a relationship between one of these and another entity (created via FKA) will automatically establish a relationship between the other entity and all entities which participate in the subclass superclass relationship with that entity. This can result in the creation

of superfluous as well as even incorrect relationships. This problem is taken up and solved in the algorithm for the particular case of relationships created out of SRs (secondary relations). The solution prescribed is to ask the user to identify the correct entity or entities that partake(s) in a relationship whenever there are several candidates (primary relations with identical primary keys).

In the example shown in Figure 5.5, the entity course should only be related to the entities teacher and student.

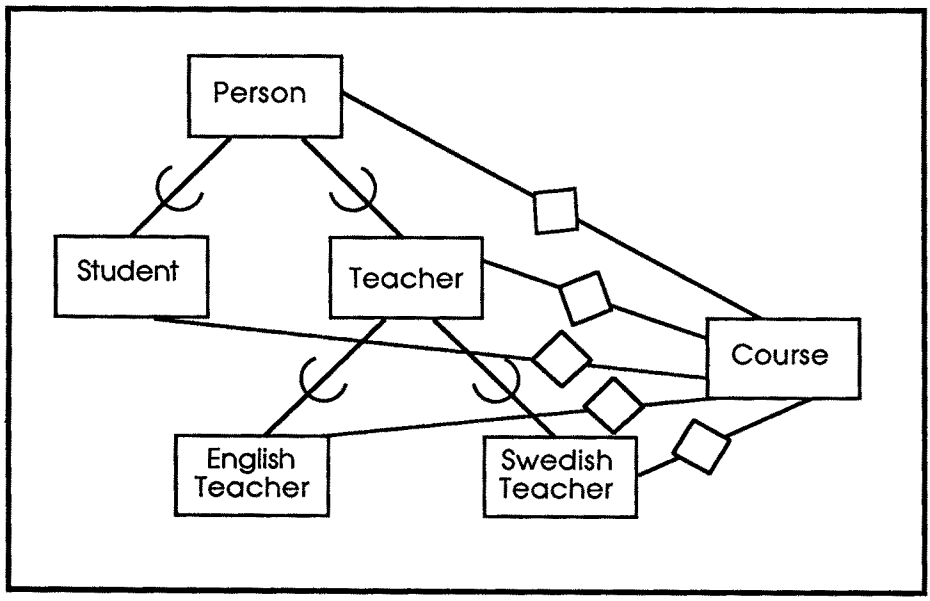


Figure 5.5 Establishment of superfluous relationships

Proposed solution:

This can be resolved by asking the user to specify which of the super and subclass entities are related to the other entity through an FKA attribute. The questions will be almost identical to the ones asked in step2 whenever there exist SRs that have attributes in their primary keys which match the primary keys of several PRs.

Figure 5.6 below shows the correct conceptual schema.

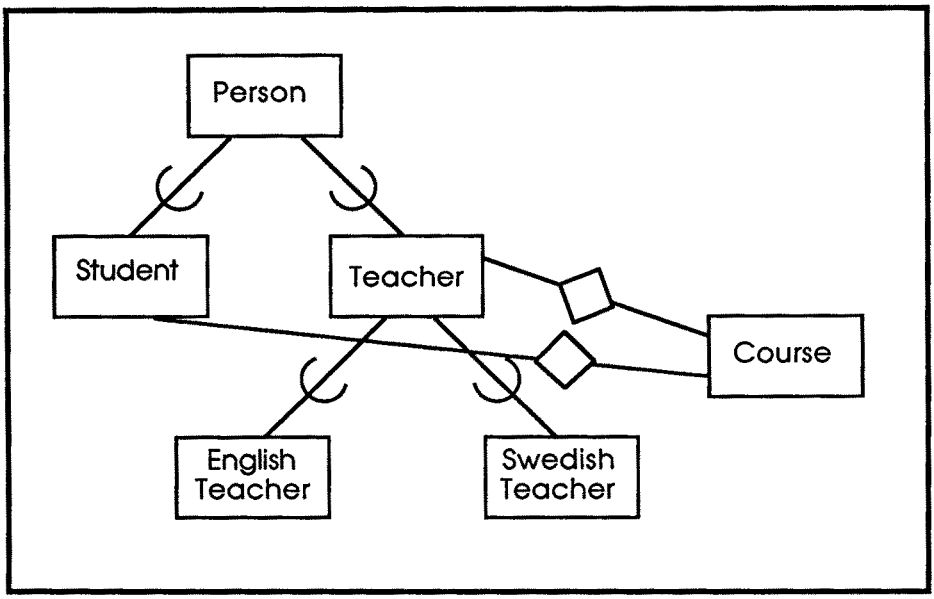


Figure 5.6 The correct schema without superfluous relationships

(4) - Creation of several new entities from one primary relation.

Problem description:

Sometimes, a primary relation (PR1 or PR2) should give rise to two or more entities. Navathe's algorithm cannot handle such cases.

Example of such a case:

country(Name,Currency).	**Name** is a primary key, **Currency** is a cand. key.
inflation_rate(Currency, Year,Rate).	**Currency** and **Year** constitute the primary key, **Rate** is a none key.

This should produce the model shown in Figure 5.7 below:

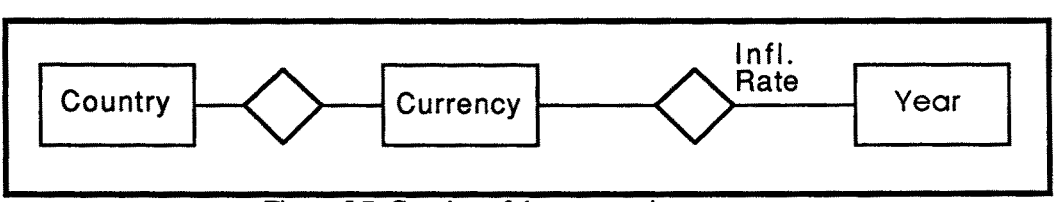


Figure 5.7 Creation of the extra entity currency

Instead MAKEMODEL produces:

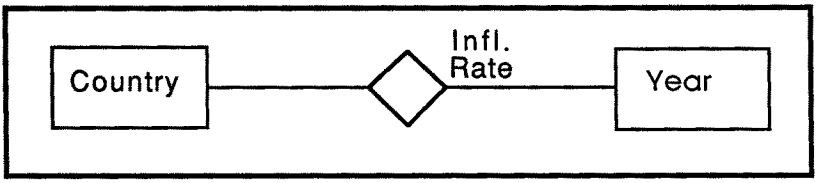


Figure 5.8 Faulty model without the entity Currency

Proposed solution:

An enhancement to the algorithm, in order to handle such cases, could be made in the following way. For each attribute in all candidate keys of a relation, ask the user if the attribute should be mapped to an entity. If the user answers yes, create a relation out of the attribute, and let its primary key be composed of one attribute whose name is the same as the name of this new relation. Take the attribute out of the candidate key of the first relation and add it to the none key attributes of the same relation. This should be carried out before any candidate key substitutions are made.

(5) - Candidate key substitutions to create sub/superclass relationships.

Problem description:

The candidate key substitution, carried out in step1.action1, which prepares relations for becoming entities that are to be subclasses of other entities, cannot handle the situation in which there is more than one superclass to a subclass. When a relation X has several candidate keys which match the primary keys of several other relations, say Y and Z, and there is a subclass/superclass relationship between the relations X and Y as well as between X and Z, the algorithm is unable to establish both of these connections.

example:

piano(P#,F@,I@).	P# is a prim. key, F@,I@ are cand. keys.
instrument(I#,**weight**).	I# is a primary key.
furniture(F#,**length**).	F# is a primary key.

After establishing a sub/super class relationship from **piano** to **instrument**, the candidate key I@ becomes the primary key of **piano** and P# becomes a candidate key. At this point there is no longer a possibility of establishing a connection between piano and furniture.

Proposed solution:

Instead of replacing the primary key of a subclass relation with one of its candidate keys (the one which matches the primary key of one of the relations which has a superclass relationship to this relation), the primary key of the superclass could, in some way, be included in the primary key of the subclass. The matching candidate key could then be deleted from the relation which is to become the subclass entity.

After the proposed candidate key substitution, the **piano** relation from the example above would be: **piano(P#,I#,F#)**. Figure 5.9 (below) illustrates the correct schema.

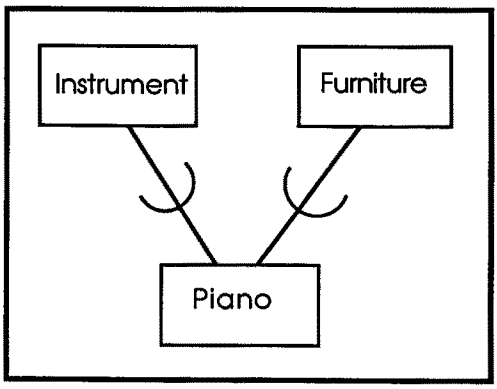
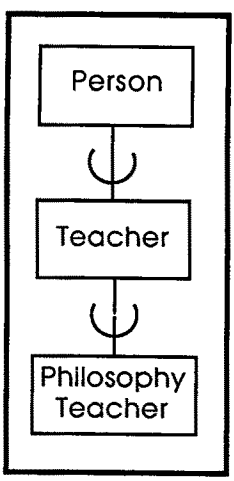


figure 5.9 An entity with two superclasses

The following is an example involving two levels of sub/superclass relationships.

person(ss_num) — **ss_num** is a primary key.
teacher(teacher_num,ss_num) — **teacher_num** prim key, **ss_num** cand.key.
phil_teacher(p_num,teacher_num) — **teacher_num** is a candidate key.



After the proposed candidate key substitution, the relations from the example above would look like this:

PERSON(SS_num)
TEACHER(Teacher_num,SS_num)
PHILOSOPHY_TEACHER(P_num,Teacher_num)

The resulting conceptual schema is shown in Figure 5.10 to the left.

Figure 5.10
Two levels of sub/superclasses.

(6) - Relationship from and to the same entity.

Problem description:

The algorithm is not equipped to express the situation where an entity has a relationship to itself. An example of this is a relationship distance which connects two cities together. This should produce the conceptual model shown in Figure 5.11.

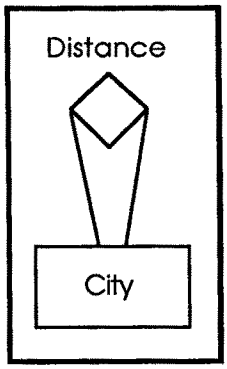


Figure 5.11
Relationship between one entity

Proposed solution:

One solution to this problem is to supply the algorithm with two relations on input, one, the relation representing the entity, the other a relation whose primary key contains the primary key of the first relation two times. For the example above the relations supplied would be: **city(city), distance(city,city)**. A relationship (distance) would then be created between two entities (city and city), but since only one such entity exists, the relationship actually connects the entity (city) with itself. This is an unnatural way of solving the problem.

Another way to represent this scenario is to have a relationship connecting two entities that have a subclass relationship to an entity. Unfortunately, the relation which is to become the relationship between these two entities will still have to contain two identical attributes in its primary key. The reason for this is that relations that get mapped to subclass entities have the same primary keys as the relation which is to become their superclass entity. Fig. 5.12 shows the resulting conceptual model.

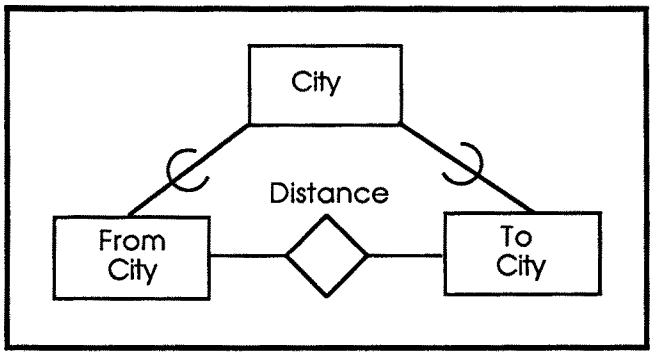
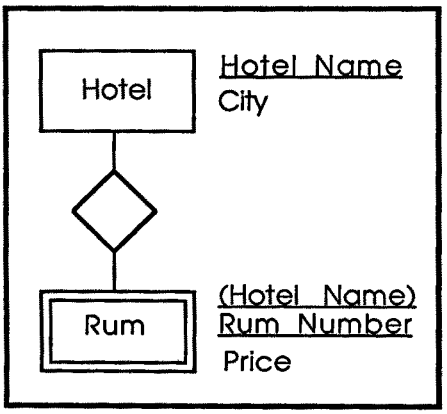


Figure 5.12 Relationship between two subclasses of the same entity

(7) - Representation of inheritance of attributes by weak entities.

Problem description:

Since a weak entity is ID dependent on another entity, it should inherit the attributes of the entity on which it is ID dependent. This point seems to have been overlooked by the authors of the algorithm and is thus not taken into consideration by MAKEMODEL.



Hotel_Name which is the identifying attribute of the entity Hotel, is inherited by the weak entity Rum. This inheritence is implicitly contained in the conceptual representaion of the ID dependence relationship produced by MAKEMODEL and should not be explicitely represented as the identifier of the weak entity Rum.

Figure 5.13 Attribute inheritance

Proposed solution:

In a similar fashion as inherited attributes become deleted from entities that have subclass relationships to other entities, inherited attributes of weak entities should also be deleted.

(8) - Establishment of faulty connections between relations.

Problem description:

The candidate key substitution, carried out in step1.action2, which establishes a connection between a relation whose candidate key is included in the primary key of another relation with that relation, can create faulty relationships between entities by establishing unnecessary connections between relations.

The following example illustrates such a case:

city(cityname).	**cityname** is a primary key.
citypopulation(cityname,year),	**cityname** and **year** compose the prim. key.
country(countryname,cityname).	**countryname** is the primary key, **cityname** is a candidate key.

After candidate key substitutions have been made, the relation citypopulation is altered to be:
citypopulation(countryname,year).

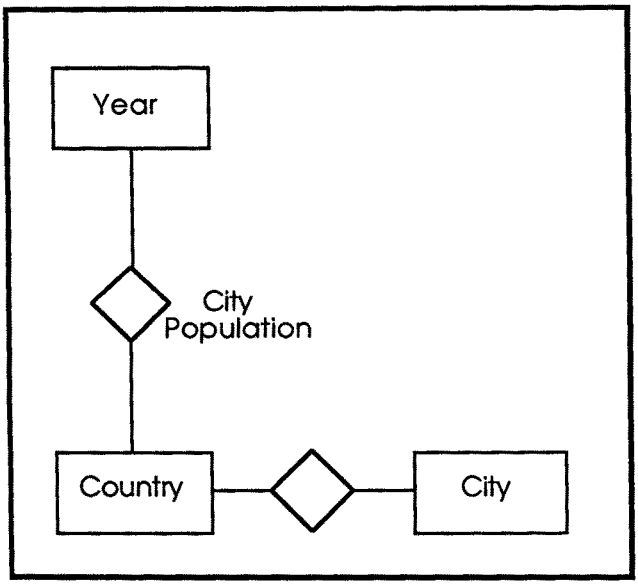


Figure 5.14 Faulty schema due to candidate key substitutions

After the candidate key substitution shown above has been carried out, a connection between the relation country and the relation citypopulation is established. Citypopulation becomes an SR2 relation giving rise to a relationship between the entity country (created from the relation country) and year (an entity created out of the attribute year in citypopulation). The entities city and country become linked through the FKA cityname in the relation country.

This results in the faulty model shown in Figure 5.14 to the left.

Without the faulty candidate key substitution a different conceptual model would be produced as shown below.

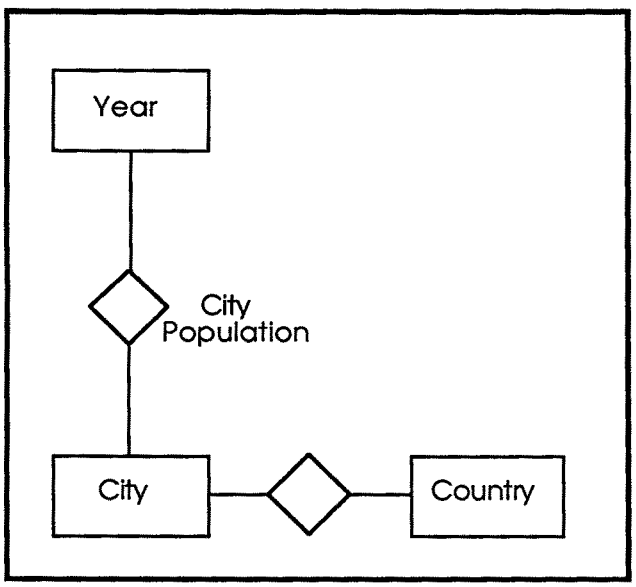


Figure 5.15 The correct schema

If the attribute cityname in country had been a none key attribute instead of a candidate key, a connection between country and citypopulation would never have been established. The existing connection between city and citypopulation would have been exploited, giving rise to the relationship citypopulation connecting the entity city with the entity year. City and country would still have been related through the FKA cityname.

This would have resulted in the correct model shown in Figure 5.15 to the left.

Proposed solution:

A more discriminate use of the candidate key substitution carried out in step1.action1 would eliminate some of the unnecessary connections which may be created between relations.

Before replacing an attribute in the primary key of a relation by a primary key of another relation, the primary keys of the remaining relations in the database should be examined. If a primary key of one of these is found to match the attribute which is under consideration for replacement, it is an indication that there already exists a connection between the relation for which there is an attempt to define a connection and another relation and the replacement should thus not be carried out.

(9) - Incapability of connecting an FKA to an SR relationship.

Problem description:

An FKA relationship cannot be established from an entity to a relationship created out of an SR2 relation. The reason for this is that since an FKA relationship is established by finding a primary key of a relation which matches a none key attribute of another relation, the primary key can only contain one attribute. This rules out SR relations.

Proposed solution:

For each KAG attribute in an SR2 relation (that is to say, attributes, found in the primary key of relationships, which do not constitute the primary keys of PR1 relations), a database search can be conducted to look for PR1 relations whose NKAs contain these KAGs. This establishes a type of an FKA connection between an SR2 and a PR1 relation. These FKA relationships would, thus, serve to identify the entity to which the KAG attribute refers and to connect that entity to the appropriate relationship.

Figure 5.16 illustrates an example of this situation as handled by the proposed solution described above. In this example the entity child had an FKA connection to the relationship marriage. This can be seen in the database relations shown below.

MARRIAGE(Wife_ID,Husband_ID,Family_ID)
WIFE(Wife_ID)
HUSBAND(Husband_ID)
CHILD(Child_ID,Family_id)

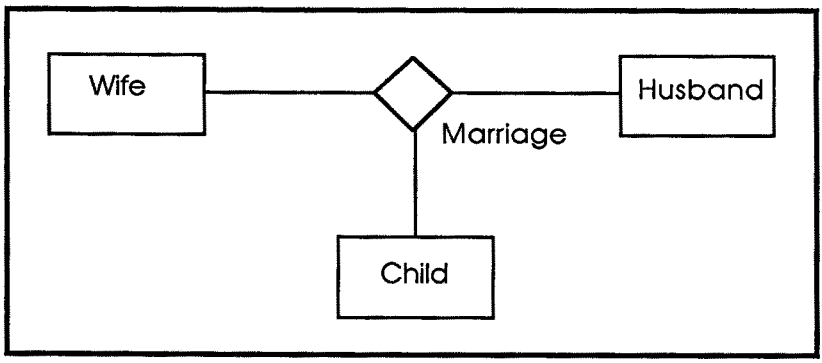


Figure 5.16 Connecting Child to Marriage via an FKA

(10) - Inappropriate entity names created from attributes.

Problem description:

Entities created from the unaccounted for (KAG) attributes in the primary key of an SR2 relation can get names which are unsuitable for an entity.

example:
The following database:
SURGEON(SS_num).
WORKS_IN(SS_num,Hosp_name).

Would give rise to the schema shown in Figure 5.17.

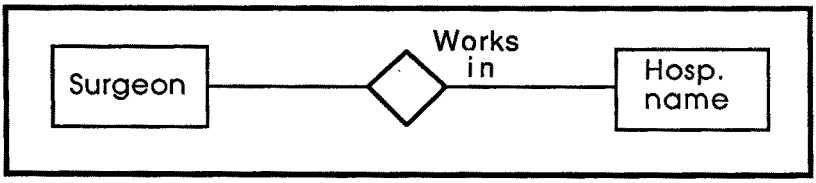


Figure 5.17 Inappropriate enttity name "Hosp name"

Proposed solution:

Consult the user. Whenever a new entity is to be created from an attribute found in the key of an SR2 relation, ask the user to specify the name the new entity.

6 CONCLUSION

There appear to exist several serious flaws in the method presented by Navathe and Awong. Many, are fundamental errors which seem to have been overlooked while constructing the method.

Most of these problems can be solved with varying degrees of difficulty. To solve the problem of having to have identical names for identical attributes, a radical change must be made to the algorithm. Solutions for each of the other ten problems outlined in the previous section have been recommended and many of these have been implemented in a revised version of the program entitled 'FIXED MAKEMODEL'. The fact that the problems can be solved, does not, however, trivialize their severity.

Most of the recommended solutions require a considerable amount of user interaction. The result is that, to a large degree, it is the user who ends up solving the problem. Such a solution is in direct opposition to the nature and purpose of the method in question, namely that of mapping a database to a semantic model by the most automated means possible, which is to say, by keeping the amount of user interaction to a minimal level.

Constructing a semantic data model from an ordinary database, often requires the presence of a certain amount of semantic information which is not, ordinarily, contained in the

database. An algorithm which is to carry out this mapping process must obtain this information in some way. The information can either be supplied by a user or derived by other means. Generally, some combination of both, user input and mechanical generation, is used. The ratio of these two means, called for by the method, determines, to a large part, the quality of the system produced. This ratio is (as argued above) unsatisfactory in the method under examination.

7 RECOMMENDED FURTHER RESEARCH

Another approach for achieving the same goal of mapping a relational database to a semantic model using a domain independent algorithm is one recommended in [2]. The method, presented in this paper, makes use of inclusion dependencies to obtain the information necessary to carry out the mapping. Inclusion dependencies are expressions consisting of two parts: a set of values and a subset of this set. Each part contains a reference to a relation as well as to an attribute of a key or none key of that relation. Inclusion dependencies, thus, function to express connections between the relations in a database. These connections are then used by the mapping process to give rise to various types of relationships between the entities of a conceptual model.

There are two noteworthy advantages to this method. The first is that by making the information supplied by inclusion dependencies accessible to the mapping algorithm, the problem of creating entities and relations becomes negligible. This alleviates obscurity and incoherence in the structure of the algorithm, (these characteristics are present in Navathe's method which relies on finding relationships by changing names of key and none key attributes, as identical concepts must have identical names). Instead, it renders the translation process easy, clean and accurate. The other and more interesting advantage of this method concerns the aspect of automating the generation of inclusion dependencies. In order to achieve this, the algorithm must be supplied with the instances of the database relations. The entire database is then searched for the purpose of extracting possible inclusion dependencies. The mechanical generation of inclusion dependencies allows for less user interaction while maintaining the high degree of accuracy and lucidity inherent in the method.

While the method described above is more coherent than Navathe and Awong's method, it still falls short in the aspect of mechanization. Reading the entire database in an attempt to establish inclusion dependencies can be greatly inefficient. Moreover the necessity of user interaction, to confirm ambiguous subset/set relationships, arises far too often. To solve this problem, an approach which takes domain into consideration could be examined.

Many researchers in the field of AI stress the importance of background knowledge for any agent (or process) that tries to extract semantic information from some given situation. John Sowa [5] proposes several ways of making such background information available. Among these are scripts and memory organization packets which are instances of schemata and prototypes. A script for a concept is a description of a sequence of events in a particular context. Memory organization packets (MOP) are like small scripts that link to other MOPs. A schema is a semantic description of a concept and a prototype is a description of a typical

instance of a concept. By augmenting a database with background information, the mapping algorithm is provided with semantic information relevant to the domain of the database. This gives the algorithm a possibility of obtaining the information it needs to carry out the mapping from a different source than the user. Moreover the conceptual model created, can be made into one which is richer in semantic information than ER or Extended ER models.

In line with the ideas presented above, a method for mapping a relational database, supplemented with domain information, to a conceptual model, rich in semantics, (of the type suggested by Sowa and illustrated in the schema above) is suggested for further research.

The data describing background information can be structured in a number of different ways, any one of which could be chosen to be used here. Scripts or memory organization packets, discussed above, are two possible representations. Frames, data objects consisting of a collection of slots that describe different aspects of the objects, presented by Minsky in [3] could be used to describe entities. An enhanced predicate logic representation APC (Annotated Predicate Calculus) presented by Stepp and Michalski in [6] is a typed predicate calculus with additional operators used to describe objects and general problem-specific background knowledge. A concept dictionary, structured as a semantic net, containing deep case structure of the concepts with default values for the cases, presented by Atlerman in [1] seems to be another feasible possibility. The choice of type of data representation is an important one which merits more research.

ACKNOWLEDGMENTS

I would like to thank Janis Bubenko, Erik Knudsen, and especially Paul Johannesson, my advisor in this project, for generously contributing of his knowledge in the field and for his critical comments.

REFERENCES

[1]. Alterman, R: "A Dictionary Based on Concept Coherence", *Artificial Intelligence, Volume 25 number 2 Feb. 1985*

[2] P. Johannesson and K. Kalman, "A Method for Translating Relational Schemas into Conceptual Schemas", *"Eighth International Conference on Entity-Relationship Approach"*, 1989.

[3]. Minsky, M: "A Framework for Representing Knowledge," in The Psychology of Computer vision, P. Winston, *0-Hill, New York, 1975*

[4]. Navathe and Awong: "Abstracting Relational and Hierarchical Data with a Semantic Data Model", *Seventh International Conference on Entity- Relationship Approach, 1987.*

[5]. Sowa, J.F. : Conceptual Structures: Information Processing in Mind and Machine, *Addison-Wesley Publishing Company, Reading Massachusetts, 1984.*

[6]. Stepp and Michalski: "Conceptual Clustering of Structured Objects: A Goal-Oriented Approach", *Artificial Intelligence, Volume 28 number 1 Feb. 1986*