Jens Andresen & Torsten Madsen

IDEAS for dynamic research oriented data recording systems

Paper presented at the 21th Jahresgatung der Gesellschaft für Klassifikation, Potsdam March 12-14, 1997.

The paper was rewritten after presentation, but never published. It appears as it was in May 1997.
Torsten Madsen 2-10-2003

Abstract

Recording of data for research purposes is problem oriented, and hence of a temporary nature. Investments in the development of a relational DBMS to record complex multidimensional data is seldom seen due to the enormity of the task balanced against the short life expectancy of the system. Data recording systems used are, however, often too simple to do justice to the complexity of data. This paper looks at the possibilities of creating a relational DBMS that can be dynamically customised at user level to different recording needs.

Introduction

In the early days of computing a very optimistic trend based on an inductive, positivist research model appeared in archaeology. It claimed it was possible to create an objective description of reality based on a finite set of descriptive elements. A standard approach for such a description was developed in the fifties and sixties with Jean-Claude Gardin as the earliest and most notable proponent (Gardin (1958), (1967)). Further, it was assumed that given a generalised, sufficiently detailed descriptive system, it would be possible to store an objective symbolic representation of archaeological materials on computers once and for all, and that given this computer based objective description, it would be possible to implement statistically based automatic classification procedures (Borillo & Gardin (1974); Chenhall (1965), referenced in Scholtz & Chenhall (1976); LeMaitre (1980); Voss (1967)).

These ideas of global solutions to computer based storage and processing of extensive descriptive bodies of data soon died out. Both on the practical and the theoretical level they proved to be untenable (Audouze & Leroi-Gourhan (1981); Cleziou et al.(1991); Hill & Evans 1972; Scholtz & Chenhall (1976)).

By the mid seventies archaeology in general had realised that the inductive procedure was not a satisfactory solution. Instead various deductive approaches were suggested, and it was maintained that “Data categories for observation, and conventions for recording data cannot be chosen independently of problem orientation.” (Scholtz & Chenhall (1976): 92).

The interest for databases in archaeological research dwindled, and whenever they come into use to day, it is generally on a modest scale as *ad hoc* recording solutions within specific projects.This is in full agreement with the current view of data modelling being specific and problem oriented. There is, however, a serious drawback to this simplistic approach to data recording. Archaeological data are very complex with a clear multidimensional structure, yet almost all recording systems are simple, and one-dimensionally structured.

Modern relational DBMS are able to handle complex, multidimensional recordings, but it can be time-consuming, tedious work to set up complex database applications, and only few archaeologists have the knowledge of how to do it properly. It is a further problem that the traditional approach to database design leads to a situation, where most changes in the conceptual design induce changes in the logical design and the user interface.

The latter problem may be illustrated through the table in Figure 1. It outlines how conceptual elements, logical elements, and elements of the user interface

Conceptual	Logical	User interface
Entities	Tables	Forms
Domains (Variables)	Columns	Fields
Tuples	Rows	Set of field values
Entity relationships	Linked tables	Linked forms

Figure 1: Correlation of conceptual, logical (formal and informal) and user interface elements using the relational model.

correlate when using the relational model. The entities defined are represented through tables, and are presented to the user through forms; the attribute of an entity are represented through a column in the table, and presented to the user through fields in the form; the instances (collectively called tuples) takes up the rows of the tables, and are available to the user as allowed values of the fields; relationships between entities are represented as primary and secondary keys in tables, and as linked forms to the user.

Following this approach, for each entity in the conceptual model, at least one table has to be created in the database and links to other entity tables have to be established, and for each variable thought necessary to describe the entity, a column has to be added to the entity table, and all this has to be reflected in the associated form.

The IDEA project (Integrated Database for Excavation Analysis)

The most recording intensive activity in archaeology is excavation. It is also the area, where most database applications has been made, with just as many different solutions as attempts.

Some years back we argued (Andresen & Madsen (1992)) that it should indeed be possible, using relational database technology, to create a database that would accommodate widely differing recording systems within the same logical structure, using the same generalised user interface.

Based on the concepts of that paper, and with financial support from the Danish Research Council for the Humanities, we have been working for the last three years on such a system called the Integrated Database for Excavation Analysis or IDEA for short (Andresen & Madsen (1996a), (1996b)).

The intended objective of the project was to create a database for the recording of excavation information, where the definition of structural and descriptive elements may take place as an ongoing process during data recording. In other

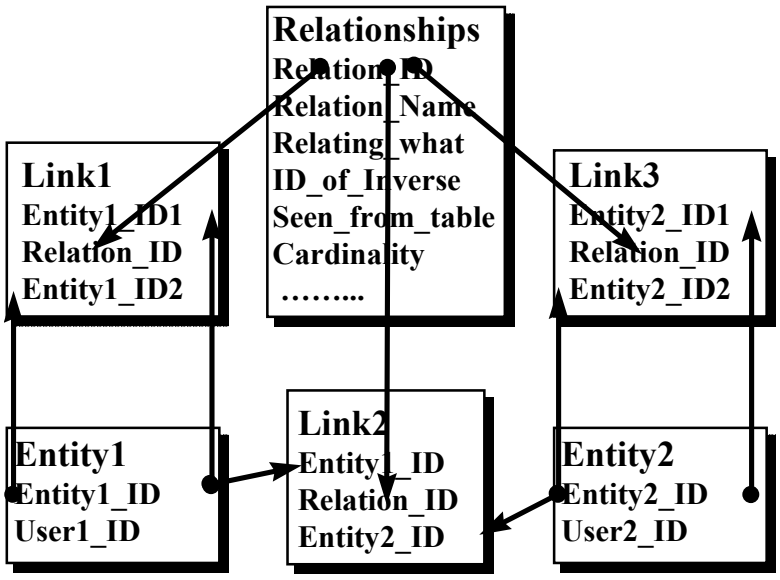

Figure 2: Basic table structure of the IDEA data base.

words, we wanted to create a system, where the user dynamically could change the specifications of recording in response to the kind of object excavated, the nature of data uncovered during excavation, and the analyses of data recorded.

Our first concern was with the basic entities. There is an obvious tendency in archaeological database applications to separate too many and too specific entities. One reason for this is that people are guided more by the variables they feel are necessary to describe the entities, than the logical substance of the entities themselves.

Our own point of departure was a rather academic analysis of the logical structure of excavation data. We separated three basic information entities, which we may name Contexts, Finds and Constructs.

Contexts are the building blocks of archaeological excavations. It is what you observe as physically existing during excavation. It is deposits, fills or building structures separated through their physical appearance. There need not be any understanding of what they represent, only that they are there.

Finds are what we separate from the contexts, stick a label to, and bring with us home. In this sense finds are just a subset of contexts, and may be viewed as such, but to archaeology it is a most important subset representing the cultural leftovers from past societies that we keep for the future.

Constructs are interpretative statements that we associate with Contexts and or Finds.

In addition to these three basic archaeological entities we have also separated two documentary entities: Drawings and Photos.

The existence of these five universal entities in recording of archaeological excavations is our basic axiom. If you do not accept it, if you can argue that there are other basic entities, then our system as conceptualised will not suffice. Otherwise, we believe, we have been able to create a system, where all specific

wishes for structure and content of a particular recording system is a matter of user customisation.

A critical point then, when working with a relational model, is the definition of universal entities. It is important that the entities defined are really basic to the problem at hand, that the tables representing them contain their identification only, and that the links between different entities are as general as possible, allowing for a flexible definition of relationships. The tables representing the entities we have defined contain only a unique identification number supplied by the system, and an identification number supplied by the user.

All entities are likely to be complex internally. That is, you may find that any tuple of an entity exhibit one or more relationships to one or more other tuples within the same entity. To handle this we use a link table to hold the ID's of the tuples to be linked (Figure 2). The link table also holds a reference to the content of a table describing the nature of the relationship between the tuples. This table holds information on the nature of the relationships. Relationships are always two-sided, and rarely symmetric. This means that we have to store both the relation between *a* and *b* and the relation between *b* and *a,* and keep track of what is the inverse of what, hence the field *ID_of_Inverse*. During data entry we take advantage of this information. Thus when a user sets a relationship between two tuples, the system automatically locates the inverse relationship and fills it in.

For each link table created at least one pair of relationships has to be established. These relationships are by definition unique to the individual link tables as they bring specific meaning to the linked pairs of tuples.

Link tables are also established between all entity tables pairwise to allow for many to many relationships between tuples from different entities. Whether these tables are to be used as many to many tables, one to many tables, or not used at all, is entirely up to the user. The same goes for constraints on individual entities. (e. g. you could demand that it should not be possible to record a find unless it has an *owned by* relationship to a context). Exactly how you wish to have the basic entities relate to each other, can be modelled through a special customisation form, and will subsequently be enforced by the system through constraints on the user interface.

As mentioned previously, the entity tables contain no descriptive elements at all. These are all to be found in a number of associated tables, the most important part of which may be characterised using the four well known questions: who, where, when and what. *Who* covers the recording of actions carried out by persons during excavation and in connection with post-excavation work. The recording includes what they did, and when they did it. *Where* is basically the spatial position. This is normally recorded in a Cartesian co-ordinate system defined locally for the excavation. *When* refers to the archaeological dating. *What* refers to the classification. In addition to the four main categories above we have also made free texts and Binary Large Objects available as descriptive elements.

In the following we will discuss how we cope with the descriptive elements, and will use the classification part of IDEA to demonstrate this. It is important

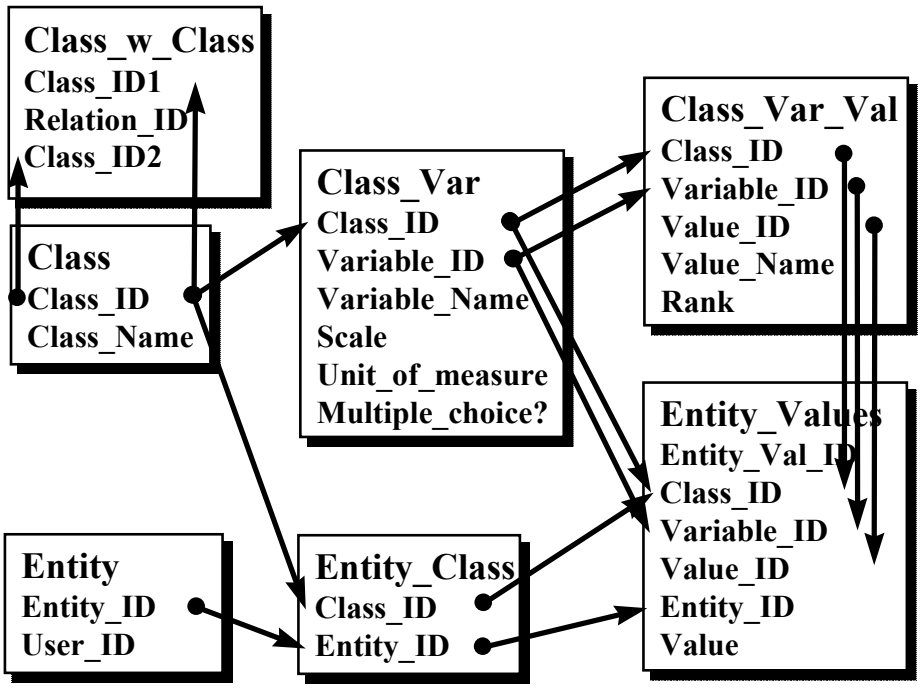

Figure 3: Basic hierarchical structure for a classification and description system.

to remember that the design of the database must be independent of whatever structure and content of the classification and description system the user chooses. We have to set up a design that will accommodate different classification and description systems simultaneously, and have no limitation to their number and structure. We may term this a meta-structure for classification and description.

Basically we have to allow for any number of classes to be defined and ascribed to entity instances, each having any number of variables, where each variable has a scale: either nominal, ordinal or interval/ratio. If the scale is nominal or ordinal we have to be able to store a set of alternative values, and for the nominal scale we should allow for multiple choice, while for the ordinal scale we should know the order of the values. For interval/ratio scale we should store a value together with the unit of measurement of this value.

It is imperative that we store the definitions of the classification and description structure in the database. It is the only safe way to ensure coherence and integrity of recordings. To the user this means that no recordings can be made before definitions of classifications and description variables has been carried through, using a special definition module. These definitions can be changed and expanded at any time with the one exception, that you cannot delete classes or description variables that has already been used in a recording. The general structure of a description system as outlined appears to be basically hierarchical with class over variable over value. So initially we have set out with a simple three table structure to represent this hierarchy (Figure 3).

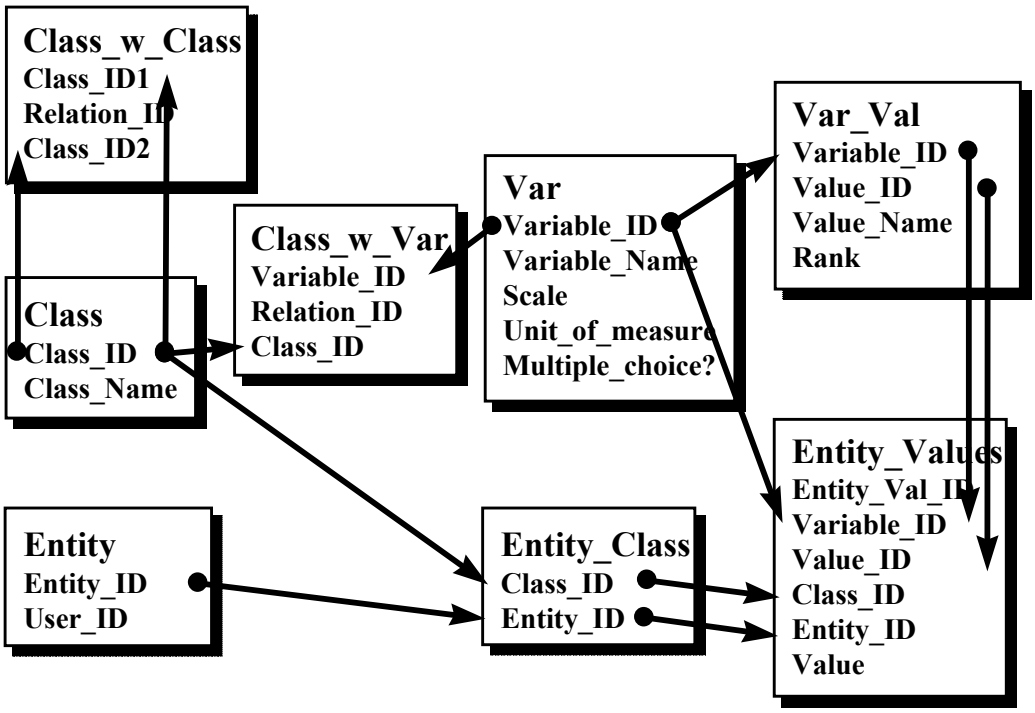

Figure 4: Classification structure with independence between variables and classes.

The table *Class* contains class name together with a unique ID, and nothing else. The table *Class_Var* linked to table *Class* on *Class_ID* contains a variable name together with a unique ID, the scale of the variable, plus a unit of measurement, and an indication for multiple choice if applicable. The table *Class_Var_Val*, linked to table *Class_Var* on *Class_ID* and *Variable_ID* contains a value name together with a unique ID, plus Rank information if applicable.

In order to be able to operate with a proper classification scheme, and not just a flat structure of parallel classes, we have added a link table combining classes with classes. This table allows in principle any structure in the core tables, but in the current version we have implemented hierarchical structures only. These are fully supported during both data entry and searches for data. As with other link tables containing the field *Relation_ID* there is direct reference to the table *Relationships,* where the nature of the link is described.

Recording a class is basically a question of pointing out which class or classes should be linked to a chosen instances. The information goes into a table *Entity_Class* connecting the table *Entity* with the table *class*. No duplicate data are entered, and all changes in both the naming of classes, and their internal structure, will have immediate effect on data already recorded.

Recording a value is very much the same as recording a class. For Nominal and Ordinal scale data, a link is established between entries in table *Entity_Class* and table *Class_Var_Val* using the table *Entity_Values*. This table is also used for the recording of interval and ratio scale variables, but here the actual value is written into field *Value* of the table.

By now, if not before, it should be obvious, why we can change content without changing physical structure. All actual recording of data is placed in the two tables *Entity_Class* and *Entity_Values*. Here we record only the combination of a class, a variable and a value. For each value, of each variable of each class of each entity tuple there will be a record in the database, and consequently for each entity tuple there may be many records, and not just one. All recordings are exactly identical in structure irrespective of the actual classification and description scheme.

When we began to use this design (Figure 3), we realised that it had a potential flaw. You will note that all variables will be unique to a particular class. No variables can be shared by two classes. This is logical if the one class is drawn from a classification of say swords and the other from a classification of pots, but it is certainly not logical, if say the rim diameter of a pot of type A should not be the same variable as the rim diameter of a pot of type B.

A remedy for this problem is to view the variables as in principle independent of the classes and establish a many to many link between them (Figure 4). We can see that apart from the new table *Class_w_Var*, the field C*lass_ID* has disappeared from the tables *Var* and *Var_Val* (formerly named *Class Var* and *Class_Var_Val*) making these fully independent of the *Class* table. It is obvious that users can now assign the same variable to widely different classes, even if it would not be meaningful to make a comparison on that particular variable, say length applied to swords and brooches. It now becomes the users responsibility to create and assign variables in a meaningful manner, say length of brooches and length of swords instead of just length.

Neither of the two solutions are fully satisfactory, however, because they do not take the user implemented classification structures into consideration. In the first, variables are by definition unique to each and every class created. In the second, it is left to the user to create a proper concordance between the classification structure defined and the associated descriptive variables. Ideally, the system should keep track of the hierarchical classification structures implemented, and let defined variables be shared according to these structures. This means that:

- All variables defined for a class at any point in the hierarchy must be shared by all other classes below that point.
- Definition of a new variable at any point in the hierarchy, should automatically be followed by a process of inheritance down the hierarchy from that point, so that the variable becomes shared among all lower classes.
- On deletion of a class or a variable in the hierarchy, it should be optional if the variables should be retained in the lower classes, or if the system should perform a cascade delete down the hierarchy on some or all involved variables. This cascade delete should automatically stop if it meets a class, where an actual recording on the particular variable has taken place.

What we are looking for is really an object oriented solution to the definition and maintenance of classification and description systems.

The way we have dealt with the classifications may be seen as an example of how we have approached the handling of descriptive attributes of entities in

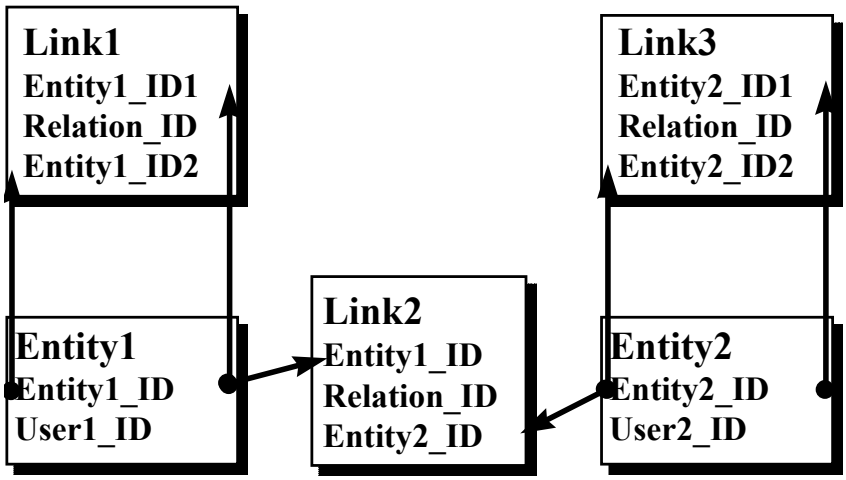

Figure 5: Abstraction from the current IDEA structure.

general. For each basic type of descriptive attributes needed we have created a design, where the user can define the specific structure and content of the descriptions, and then subsequently apply them in the actual recordings. All customization can be done at user level without changes in the database design being necessary.

Beyond the IDEA. A look at future developments

IDEA is a system developed with specific reference to the recording of excavation information. Due to its generalised design structure, it might have been turned into a recording system for a lot of other things as well. Many colleagues, with whom we have discussed the system, have immediately seen other possible areas of application. By simply changing texts appearing in the user interface, it would be possible to change the system from a dedicated excavation recording system to be a system dedicated to the recording of something else.

This of course was never in our minds, but it would not be difficult to change the design of the system so that users could redefine the labelling of the forms. Together with a few other necessary changes of design this would provide a general purpose recording system supporting up to five basic entities.

Five entities would be the limit, however, and that is really a weakness of our design. User customisation at entity level is not possible. Recalling the table of the traditional dependency between the conceptual level, the logical level and the user interface (Figure 1), we have successfully tangled the problem of different variables being bound to different columns in tables, but we have not solved the problem of different entities being bound to different tables. Can we solve this problem as well? On closer inspection it turns out, that the problem is not just with the tables, it is also and perhaps more importantly with the forms.

Since each entity has to be presented through a form, and since all forms representing entities should potentially be open at the same time due to constrained linking - that is, a tuple in one entity cannot exist unless it is linked to a tuple in

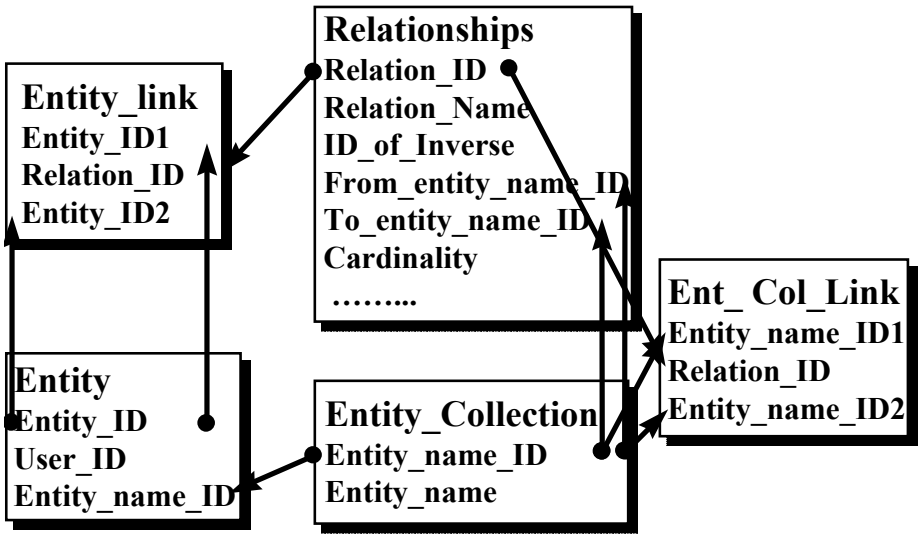

Figure 6: Generalised design with support for multiple user defined entities.

another entity - we have no other choice than to create one form for each entity. A new entity thus cannot be created, unless a new form is created as well, which means physical changes to the database. Or rather it meant physical changes to the database, because in the recent Access 7.0, which is the development software we use, a new feature has been added, which alleviates this problem. Forms can now be cloned so that you need only create one form of a particular type. Whenever you need a new form of this type, you can create it on the fly as a copy of the original one, and then set its features programmatically, or even change its appearance.

So the focus is back on the tables. Can we create a design, where the observation entities are not dependent on being represented by a separate table? The answer is yes, and the solution is in fact quite simple.

If you look at this abstraction from the current IDEA structure (Figure 5), showing two entities with their internal and external link tables, you will note that there are only two different types of tables: entity tables and link tables. Tables holding different entities are identical, and the same applies to all link tables, whether they set up links internally between tuples of the same entity, or connect tuples of different entities.

If we introduce the concept of *entity collection*, we may collapse everything into one single structure (figure 6). This structure will contain one entity table, one link table and one table containing the collection of entities. This structure will apply regardless of the number of entities defined.

In our current design in the table R*elationships* we have two fields that are used to keep track of the domain of a relationship (figure 2). The field *Relating_what* contain names supplied by the program, indicating which link table a relation apply to, and the field *Seen_from_table*, contain names supplied by the program, indicating from which of the two entity tables joined by the link table, the relation should be seen.

This construction will not work, of course, if we place all entities in one table, and all relationships between entities within one single link table. Instead we have to make use of the entity names supplied by the user in the table *Entity_collection (Figure 6)*. A relationship in a link table will always be from one named entity to either a different or identical named entity. Thus we need two fields in the table *Relationships* holding ID's of entity names: one for the ID of the entity name from where the relationship initiates, and one for the ID of the entity name to where the relationship leads. There will of course be a one to many link between the table *Entity_collection* and the table *Relationships* on these ID fields.

Finally we must allow the user to control how the defined entities should relate to each other. We do this by way of the *Ent_link_col* table establishing a many to many relationship between the entries of Entity_name_ID in the Entity_collection table. To qualify these inter-entity relationships we may use the table *Relationships* as with all other link tables. To link the classification and description system discussed above to this structure, only a few adjustments are needed.

Compared to our current design the changes in the overall design of the table structure is very small. Yet we have made no attempt to implement the changes because it turns out that they will have far reaching consequences for the design of the user interface. Hardly any form would remain unchanged, and the major part of the code behind the forms providing the functionality of the system would have to be rewritten. It is a real challenge to implement this design, but we cannot do it within the financial frames of the current project.

Conclusion

The purpose of this paper has been to demonstrate, that it is indeed possible to design a common database structure that can serve as a container for recordings of empirical observations in research, regardless of how these observations are being structured by the observer. The point of departure was the creation of a generalised recording system for a specific purpose, the recording of archaeological excavations. The specialisation that this has brought to the actual system created is obvious, and has been intentional. At the same time it should also be obvious, however, that through the process of creating this generalised system for excavation recording, we have come close to breaking into a much more general solution. It was not in our minds when we started, but today we do not doubt that a truly general solution to the design of a recording system for research can be reached.

There are great advantages to be gained from such a system. Thus, all efforts placed in its development will be to the benefit of all, who use it for recording, regardless of the structure and content of their recordings. Further, different recording systems can reside together, and be compared with each other to the degree they share a common structure and content, or can be translated from the one to the other. Also, it is important that different classifications and descriptions of a material can be used simultaneously. It makes it possible to compare the feasibility of different classification system, and at the same time it makes the importance of data standards debatable.