APPLIED COMPUTING, MATHEMATICS AND STATISTICS GROUP
Division of Applied Management and Computing

From Conceptual Model to End User Implementation

Clare Churcher, Theresa McLennan and Alan McKinnon

Research Report No:04/2001
October 2001

ISSN 1174-6696

RESEARCH REPORT

LINCOLN UNIVERSITY
Te Whare Wānaka O Aoraki

Applied Computing, Mathematics and Statistics

The Applied Computing, Mathematics and Statistics Group (ACMS) comprises staff of the Applied Management and Computing Division at Lincoln University whose research and teaching interests are in computing and quantitative disciplines. Previously this group was the academic section of the Centre for Computing and Biometrics at Lincoln University.

The group teaches subjects leading to a Bachelor of Applied Computing degree and a computing major in the Bachelor of Commerce and Management. In addition, it contributes computing, statistics and mathematics subjects to a wide range of other Lincoln University degrees. In particular students can take a computing and mathematics major in the BSc.

The ACMS group is strongly involved in postgraduate teaching leading to honours, masters and PhD degrees. Research interests are in modelling and simulation, applied statistics, end user computing, computer assisted learning, aspects of computer networking, geometric modelling and visualisation.

Research Reports

Every paper appearing in this series has undergone editorial review within the ACMS group. The editorial panel is selected by an editor who is appointed by the Chair of the Applied Management and Computing Division Research Committee.

The views expressed in this paper are not necessarily the same as those held by members of the editorial panel. The accuracy of the information presented in this paper is the sole responsibility of the authors.

This series is a continuation of the series "Centre for Computing and Biometrics Research Report" ISSN 1173-8405.

Copyright

Correspondence

This paper represents work to date and may not necessarily form the basis for the authors' final conclusions relating to this topic. It is likely, however, that the paper will appear in some form in a journal or in conference proceedings in the near future. The authors would be pleased to receive correspondence in connection with any of the issues raised in this paper. Please contact the authors either by email or by writing to the address below.

Any correspondence concerning the series should be sent to:

The Editor
Applied Computing, Mathematics and Statistics Group
Applied Management and Computing Division
PO Box 84
Lincoln University
Canterbury
NEW ZEALAND

Email: computing@lincoln.ac.nz

From Conceptual Model to End User Implementation

Clare Churcher, Theresa McLennan and Alan McKinnon
Applied Computing, Mathematics and Statistics Group
Lincoln University
P.O. Box 84, Lincoln University
Canterbury
New Zealand
email:churchec/mclennan/mckinnon@lincoln.ac.nz

Abstract

For a number of reasons many users are responsible for the implementation and maintenance of their own databases. While they may have the technical skills to set up data repositories, many end users lack the analysis skills to design a conceptual model which accurately reflects the subtleties and complexities of their requirements. In this paper we discuss why it is important for end users to obtain help in developing a full conceptual model of their data. We propose a number of ways to capture the most essential aspects of the model to produce a simplified design that an end user can be reasonably confident of implementing and using accurately. We discuss how the simplifications may impact on the final application. Our simplification methods are illustrated with an example of a scientific database and we also show how to represent the simplified model in both a database and a spreadsheet.

1. Introduction

Conceptual modelling has evolved considerably since the early entity-relationship models (Chen, 1976). A number of methodologies and notations have been proposed as the transition has been made through structured analysis (Yourdon, 1989; de Marco, 1979), data driven design (Jackson, 1975; Warnier, 1977) to more object oriented models (Shlaer and Mellor, 1988; Martin and Odell, 1998; Booch and Rumbaugh, 1995). There has also been considerable work in identifying patterns of data models and in applying these to a variety of problems (Hay, 1996; Fowler, 1997). However these techniques and methodologies are not designed with end users in mind. We do not wish to address the causes of failures in large or corporate wide databases, however the difficulty in successfully implementing these systems often results in frustrated end users developing numerous small systems themselves (Gunton, 1988). These applications are seldom carefully designed and problems inevitably arise: data becomes inconsistent, some queries become impossible, statistics and analyses have errors. These types of problems occur in many databases and are well documented for spreadsheets (Panko, 2001).

End users, characterised in Section 2, usually have a good feel for how their data is collected and how it might be analysed, but they seldom have the skills to represent all the detail accurately in a conceptual model. Without those skills the resulting database will probably not provide the functionality they require. Users frequently fail to anticipate the exceptions and irregularities that will inevitably arise and may impact on their application, especially as it evolves. They also tend to

make considerable use of textual comments that are very difficult to convert into a format that can support queries. Kreie (2000) investigates the effects of providing spreadsheet users with training in system analysis and design. While the design (layout, documentation and correct use of references) of the resulting spreadsheets improved, the effect on their accuracy and completeness was unclear. We maintain that good analysis skills are not gained quickly but require considerable experience. In Section 3 we explain why we would encourage any keeper of data to seek early expert advice in producing a comprehensive data model for their problem (Churcher *et al.*, 2000).

We introduce an example end user problem in Section 4, showing the user's initial attempt at storing data and the problems encountered. We also present the full conceptual model which we use to illustrate the concepts discussed in the following sections.

In Section 5 we look at pragmatic ways of taking a complex conceptual model and simplifying it so that a reasonably competent end user can implement and maintain a system that meets the most important requirements. This needs to be done with the user fully aware of the compromises he or she is making and how they will affect the flexibility of the eventual system. In Section 6 we discuss how the model can be implemented in a database or a spreadsheet. We advocate that if a full conceptual model of the original problem is developed early, a simple (but accurate) system can be developed which can grow in a controlled way as time, resources and skills become available. We discuss this in Section 7.

2. End users: Who are they?

Increasingly, end users are being required to construct their own solutions to computing problems using application software. Typically these people would not regard themselves as computer professionals. This is the concept of end user computing and has been well discussed in the literature (Panko, 1987; Delligatta, 1992; Halloran, 1993). A wide variety of people depend on end user tools for handling their data requirements. For example, small businesses who do not have the resources to employ database professionals make considerable use of spreadsheets and databases which they construct themselves. Even within large institutions many employees will value the independence and immediacy afforded by constructing their own data repositories (Gunton, 1988). Often they will extract subsets of the corporate data into their own applications to analyse and even update. The uncontrolled proliferation of end user databases of this sort has led some institutions to discourage the use of desktop database management software causing the data to end up in spreadsheets instead (Lincoln University Industry Computing Liaison Group, 1999). While the ideal world would have corporate databases that provided all the users' needs in a timely and efficient manner, the reality is that users value the control afforded by end user tools.

The end user focus is typically narrow and immediate. This is especially common in research institutions. Of necessity individual researchers have very specific requirements and the data they collect is often kept in spreadsheets designed to meet their initial analytical needs (Churcher *et al.*, 1998). Valuable data can be stored in such a way that much information is lost or is susceptible to erroneous analysis. It takes time and very specific skills to design a useful database for scientific data but the pressure to store the data and perform some quick calculations usually takes precedence over the longer term view.

3. End user databases

3.1. Problems encountered with inadequate design

Few end user systems go through any formal design process. For scientific problems, database tables or spreadsheets are often set up to reflect the way the data is collected. For commercial systems where the emphasis may be on reporting, the user may set up the data stores to reflect the layout of an essential report. Data driven design methodologies (Jackson, 1975; Warnier, 1977) start at this point, but end users seldom carry through the design process. The result is that while the immediate data storage needs may be satisfied, a number of problems may quickly arise. These can be loosely categorized into short, medium and long term problems which we summarise here and illustrate in subsequent sections.

In the short term:

> Data may become inconsistent as a result of redundancy and exceptional or missing data may not be handled correctly. This results in the initial queries or analyses being inaccurate.

In the medium term:

> New queries may be difficult or impossible as a result of storing data inappropriately e.g. in textual fields. Integration with other data sets may be difficult or impossible.

In the longer term:

> If data values that change slowly over the long term have not been carefully considered then historical data may be lost. It may be difficult to expand the database to include new information and as the data set becomes large it may be difficult to export to different software.

Hobbs and Pigott (2000) recommend that users set up their data in a way that is natural for them and then bring in expert help to convert it into a relational format. They point out that end users naturally think of their data as lists. In addition, end users frequently store a considerable amount of information in textual comment fields, often because this is how they collect the data e.g. weather conditions. While this may satisfy the immediate need of storing data, in the medium term useful information will be lost. Queries such as *show me all the counts excluding those taken when raining* will be impossible unless the weather information is appropriately categorised. As noted by Hobbs and Pigott it is very difficult to convert information stored in textual form into a useful relational format. If end users are encouraged to set up a system and populate it without some initial advice they are in serious danger of having a great deal of important information recorded in a way that is impossible to query.

Even if a full data model is developed with expert help, problems still arise. An end user is unlikely to have the skills or time to implement and maintain a complex model. In particular the more complex a model the more difficult it is to perform queries. Complex models result in databases with many tables and the number of joins involved even for a simple query can be daunting. Certainly some representative views can be set up but at this point end users become increasingly reliant on the expert and lose their independence and control.

3.2. Suggested design process

Our approach is to encourage users to seek help in developing a comprehensive conceptual model and then to extract a subset of the most important aspects. The reduced model is more easily developed and maintained by the end user who thus retains control of the application. The result is an application which is accurate and delivers the required information albeit within a reduced scope.

An expert's input early in the design process is necessary to ensure that valuable information can be retrieved and the database can evolve as required. With an expert's help the user can be encouraged to think ahead to what they might want to extract from the data they are storing or what other reports may be useful. We advocate that an initial full conceptual model should be developed with expert help as early as possible in the process. By a full model we mean one that includes all the data that a user is currently collecting and is capable of producing all the information a user requires in the short to medium term. The exercise is valuable in encouraging the users to gain a better understanding of their data and the implications of how it is stored. It encourages users to think critically about what it is they require from their data now and what they may wish to be able to do in the medium and long term.

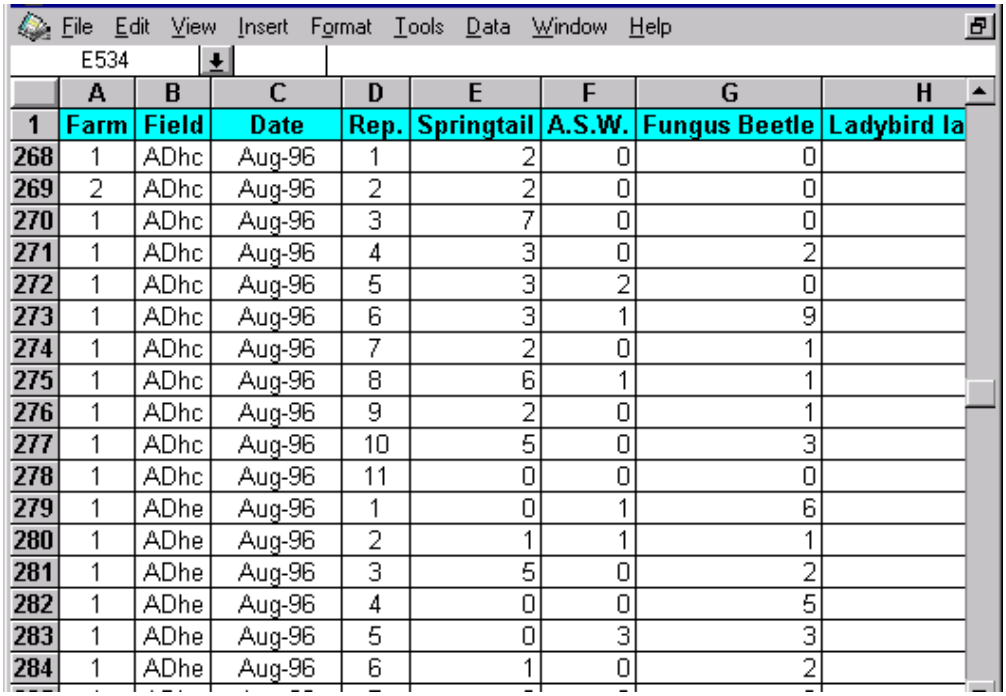
In the following sections we suggest a number of ways to simplify a model so that the user can construct a simpler system that captures their most important data accurately. The full model is essential for the user to understand the consequences of the simplifications and to be able to make informed decisions about what they wish to implement. At a later date some of the extra intricacies can be developed as skills and resources become available. We discuss this further in section 7.

4. A scientific example

As an illustration we consider an actual example of scientific data collected about populations of insects. This was a large, long term research project. The scientists would visit farms, collect samples and count the numbers of certain indicator species of insect present. Over the long term trends in the number of insects could be compared against changes in the management of farms to help understand the long term environmental effects of such changes.

4.1. Recording of data

The researchers used a spreadsheet to record their results from the field, that being the easiest and quickest solution in the short term. A section of the spreadsheet is shown in Figure 1. There was also a considerable amount of extra information kept in a textual format. This included: information about the farm (location, owner), type of field (modern arable, organic arable), soil type, weather conditions, plant cover, stock, transects (whereabouts in the field a sample was taken), a history of spraying and other field treatments and the collection method.
Help was first sought only when it became clear that a spreadsheet would not handle the volume of data being collected. However it was immediately apparent that there were other more serious flaws. There was clearly a great deal of redundant data and the subsequent inconsistencies were already evident. More importantly, the data was not kept in a way that would easily answer some of the most likely research questions.



File Edit View Insert Format Tools Data Window Help

E534

	A	B	C	D	E	F	G	H
1	Farm	Field	Date	Rep.	Springtail	A.S.W.	Fungus Beetle	Ladybird la
268	1	ADhc	Aug-96	1	2	0	0	
269	2	ADhc	Aug-96	2	2	0	0	
270	1	ADhc	Aug-96	3	7	0	0	
271	1	ADhc	Aug-96	4	3	0	2	
272	1	ADhc	Aug-96	5	3	2	0	
273	1	ADhc	Aug-96	6	3	1	9	
274	1	ADhc	Aug-96	7	2	0	1	
275	1	ADhc	Aug-96	8	6	1	1	
276	1	ADhc	Aug-96	9	2	0	1	
277	1	ADhc	Aug-96	10	5	0	3	
278	1	ADhc	Aug-96	11	0	0	0	
279	1	ADhe	Aug-96	1	0	1	6	
280	1	ADhe	Aug-96	2	1	1	1	
281	1	ADhe	Aug-96	3	5	0	2	
282	1	ADhe	Aug-96	4	0	0	5	
283	1	ADhe	Aug-96	5	0	3	3	
284	1	ADhe	Aug-96	6	1	0	2	

Figure 1: Original spreadsheet storing counts of insects.

For example

- *Show me the average counts of each insect species for each different field type over the last 5 years.*
- *Show me the counts of springtails for all fields excluding those that have been sprayed within a week of sampling.*

While the data to answer these questions had been collected, it was mostly in text and recorded in a way which could not be queried accurately. An even more serious problem was that the management type of the field was included in the coded identifier of the field thus making it extremely difficult to cope with the inevitable changes when they occurred.

4.2. A conceptual model

The problem here was that the scientists, quite understandably, had not developed a thorough conceptual model for their data. Without this they were not able to appreciate the downstream problems they were likely to encounter. This is a very familiar scenario with end user scientific data repositories.

As a first attempt at rectifying the problem a full conceptual model was developed. A portion of this is shown in Figure 2.

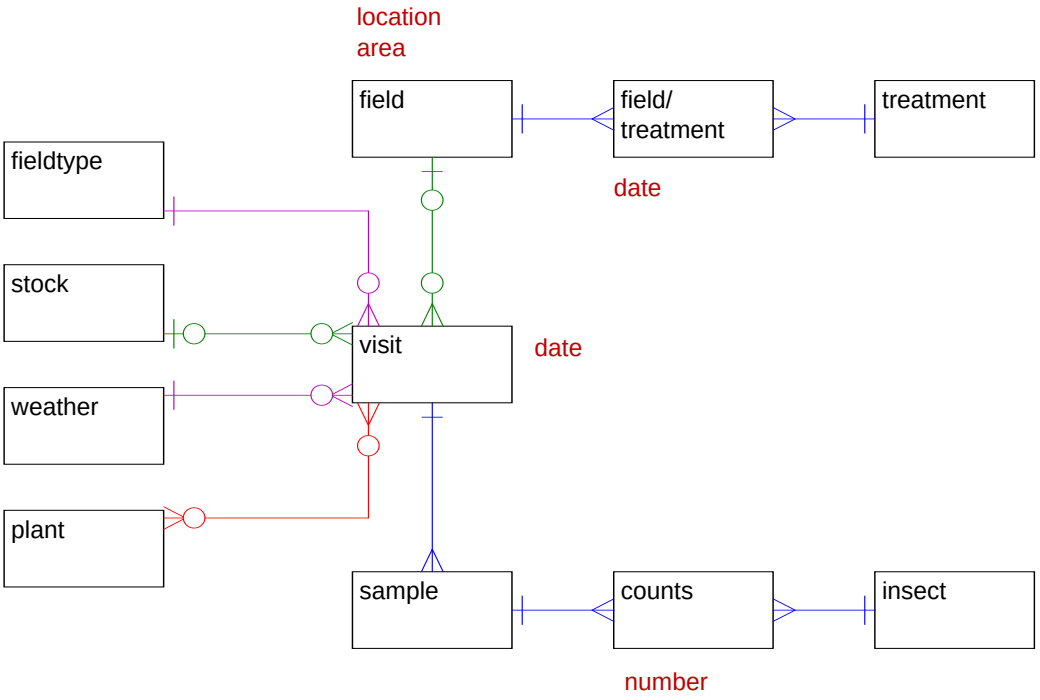


Figure 2: Part of the conceptual model for the insect data.

The exercise was very valuable for all concerned in that it clarified a number of issues previously referred to only in textual notes. For example they were interested in recording all the plants present but only the main type of stock. If the weather changed during a visit collection stopped, and most importantly the type of a field could change so should be an attribute of the visit rather than the field. Although this was fundamental to the project, the way the fields were originally coded made the change very difficult.

The users' confusion over some of these issues reinforced our belief that although end users may understand the philosophy or science behind their data they are not necessarily expert in understanding the importance of some aspects in terms of designing a database.

4.3. Implementing the full conceptual model

In this particular case the full conceptual model was developed by professional database personnel. However the users did not embrace the result with as much enthusiasm as expected. While the model in Figure 2 gave considerable flexibility and was able to cope with many of the unusual situations that occurred, the complexity meant that the users became dependent on the database professionals. This was particularly evident when new reports and ad hoc queries were required. The end users were not comfortable with queries involving numerous joins and thus lost the control and immediacy that they valued. In the next section we propose some generic simplifications which could have been used in this case to retain the main features of the model while making it more end user friendly.

5. Simplifying the model

The following steps are a summary of the ways it is possible to simplify a conceptual model.

i. Distinguish the fact and catagorising classes
ii. Represent each simple 1-many category with a constraint
iii. Represent each simple many-many category with a constraint
iv. Simplify more complex categorizing data
v. Represent the fact data

We illustrate each of these points by describing the reduction of the conceptual model in Figure 2 to the simplified version shown in Figure 3.

5.1. Distinguishing fact and categorising classes

The classes in Figure 2 can be loosely divided into two types: categorising and fact. We distinguish these by separating those classes whose data is essentially constant from those which have data regularly added. This is similar to the distinction between fact and dimension tables in a dimensional model (Kimball 1998, Adamson 1998).

Many of the classes in Figure 2 are a means of categorizing and recording information that was previously stored in descriptive textual notes (for example the *stock, plant,* and *weather* classes). Other categorizing data was previously dealt with by codes (for example *field*) or by separate columns in the spreadsheet (*insect*). Categorizing data is updated rarely if at all. Updates would mostly involve adding additional categories.

The fact data is added to regularly. In this case it is the data which is recorded during an experimental session and it is mostly to be found in the *counts* class (numbers of insects) and the *visit* class (the date and conditions of the visit).

5.2. Representing simple 1-many categories as constraints

Some categorizing data only simple descriptions of the different categories. E.g. for *weather* the requirement was for a very coarse distinction and the values *fine, overcast* and *rain* were sufficient. It is possible to capture this detail by implementing a validation constraint on an attribute *weather* in the visit class (Figure 3). Any additions or changes to these categories should be carefully considered as this could cause confusion about the data already collected.

The classes such as *stock*, and *fieldtype* are different from *weather* in that it is reasonable to expect additional values to be necessary over time. These classes could be implemented as tables to make it easier for the user to add extra values. The downside of this is that queries involve more joins and are thus more difficult for the user to develop.

A pragmatic solution for an application which is to developed, queried and maintained by an end user is to implement these category classes as constraints in the same way as *weather*. New values will occasionally require a change in the design but a competent end user will not find that difficult and the reward will be a much simpler database to implement, maintain and query.

5.3. Representing a simple many-many category with a constraint

The *plant* category is different to the *stock, weather* and *fieldtype* classes as it is necessary to record several plants for each visit. This was originally handled by recording each plant in a textual note. Another common way of dealing with this situation is for scientists to decide on an upper limit for the number of plants they will record and to add that many columns to the spreadsheet (e.g. *plant1, plant2, plant3).* This makes the finding of counts when a particular plant or combination of plants is present very awkward. For an implementation in a relational database the addition of a *plant/visit* relation is unavoidable. However using a constraint on the plant attribute for that relation can eliminate the need for a *plant* table.

5.4. Representing more complex categorizing data

The *plant, stock, weather* and *fieldtype* data were all simple in that it was not necessary to keep any information about the categories. The *treatment* class which kept data about herbicide and pesticide applications is more complicated. There was considerable information that could be collected about the treatments. Deciding on the correct way to model this becomes a choice dependent on how important the detail is likely to be. If the users can distinguish a few main categories (say less than a dozen) then a constraint is a possible solution. If they require much more information (concentrations, chemical compositions, application methods) then a more complex structure will be needed. This now becomes a pragmatic issue. How critical is this information to the overall project and is it worth the extra complexity and inevitable loss of control? The user needs to be asked quite specific questions to ensure they make an appropriate choice. E.g.

> *Is it critical for your analyses to be able to extract all records for fields with recent treatments involving a particular concentration of Pesticide A?*
> *Or*
> *Will it be sufficient to be able to identify records only by the fact that they have had a pesticide applied recently?*

The decision may well be that the simpler question will do for the time being but the way is left open to add the other information later if that is necessary (see Section 7).

The class *field* cannot be dealt with as a constraint as information about each field such as the location and size are required. *Field* also has connections to the *treatment* class and this also makes dealing with it as a constraint impractical.

While there is a great deal of information that could be kept about *insects* that was not an essential requirement of the project and as the number of insects being investigated was only about 15 it would be possible to implement that class as a constraint also.

5.5. Representing the fact data

The exercise of constructing the conceptual model highlights that there are three different types of fact that are being constantly added to: *visits, counts,* and *field/treatments*. Field treatments had previously been kept quite separately from the other data in a farm diary. The visit date had been kept as redundant data along with each count in the spreadsheet (see Figure 1) and the other attributes of the visit were kept as notes. The field type was kept in the coded field name with the attendant difficulty of recording changes. The conceptual model helps the end user see that these different types of data are similar and should be treated similarly.

It is possible to remove the *sample* class if no information is required about each sample (e.g. position, volume). The sample number can then be kept as an attribute of the counts. If data about the sample is required then a separate class will be needed to avoid redundancy and the ensuing inconsistencies.

A model incorporating the simplifications described in sections 5.1 - 5.5 is shown in Figure 3.

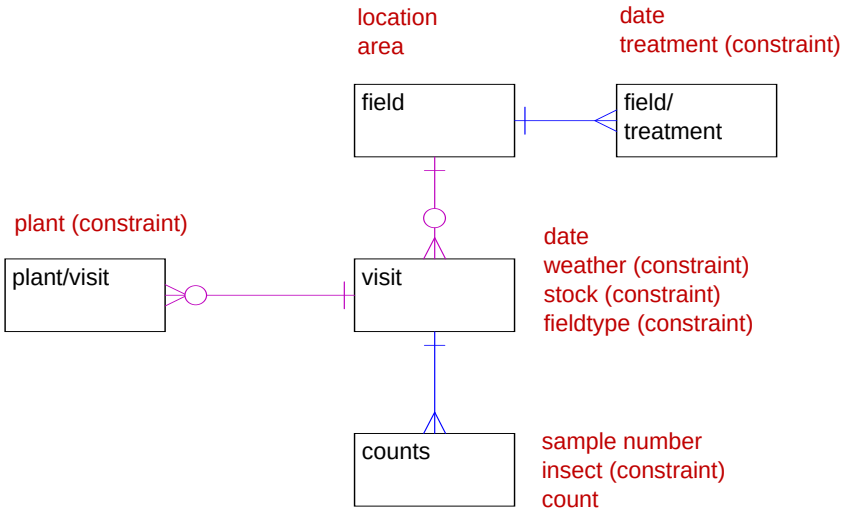


Figure 3: Simplified conceptual model

6. Implementing the simplified model.

After applying the simplification steps to produce the model in Figure 3 from that in Figure 2 we see that the number of classes has reduced from 11 to 5 and the number of relationships from 10 to 4. The complexity is clearly much reduced without losing the essential detail of the problem. We now describe how an end user might implement the simplified model in both a database and a spreadsheet.

6.1. Implementing the model in a relational database

The following list is a summary of the steps for implementing the simplified model in a relational database.

i. Create a table for each class.
ii. Determine a primary key for each table.
iii. Set up a foreign key for each relationship.
iv. Set up validations on the fields for each constraint

For this particular problem the first step involves setting up 5 tables, one for each class. Because both the *visit* and *field* tables are at the one end of a relationship it is convenient to introduce a new attribute to function as the primary key (e.g fieldID, visitID). The other tables can use concatenations of existing attributes for their primary keys: *counts* (visitID, sample number, insect), *plant/visit* (plant, visitID) and *field/treatment*(fieldID, treatment) (assuming that a particular treatment is only applied once a day). The relationships can then all be represented by foreign key constraints referring to either the *visit* or *field* relations. The final step is to implement the constraints as

described in the previous section by using the validation checks on fields appropriate to the software being used.

This is fairly straightforward for an end user with a good basic understanding of relational theory to implement. Forms can be set up to make the data entry easier. To assist with querying and reporting it might be sensible to construct a view joining all the tables on their keys and foreign keys which could then have simple select criteria added as required. This would deal with the bulk of the querying required to do simple analyses. More complex queries involving different joins (e.g. find visits which had non zero counts for both insect A and insect B) would need to be constructed at the time and may be beyond the expertise of many end users. However, this implementation is vastly superior to the original implementation in Figure 1 in keeping data accurately and allowing effective queries to be made. It will also be more easily managed and queried by end users than the full model in Figure 2.

6.2. Implementing the model in a spreadsheet

End users are more likely to feel they are competent users of spreadsheets than they are of databases McLennan *et al* (1998). Certainly, until recently users were more likely to have access to spreadsheet than database management software. Even now only the professional, or higher, versions of the popular office suites (e.g. Microsoft Office and WordPerfect Office) include a relational database. Informal discussions with user support managers (Lincoln University Computer Industry Liaison Group 1999) suggest that it is not uncommon for end users in a large organisation to be denied access to relational database management software when developing their own applications. For these reasons it may well be pragmatic for an end user to store their data in a spreadsheet.

It is possible to capture much of the simplified model, Figure 3, in a spreadsheet. Below is a summary of the steps involved in implementing a schema such as that in Figure 3 in a spreadsheet.

i. Construct a separate sheet for each class

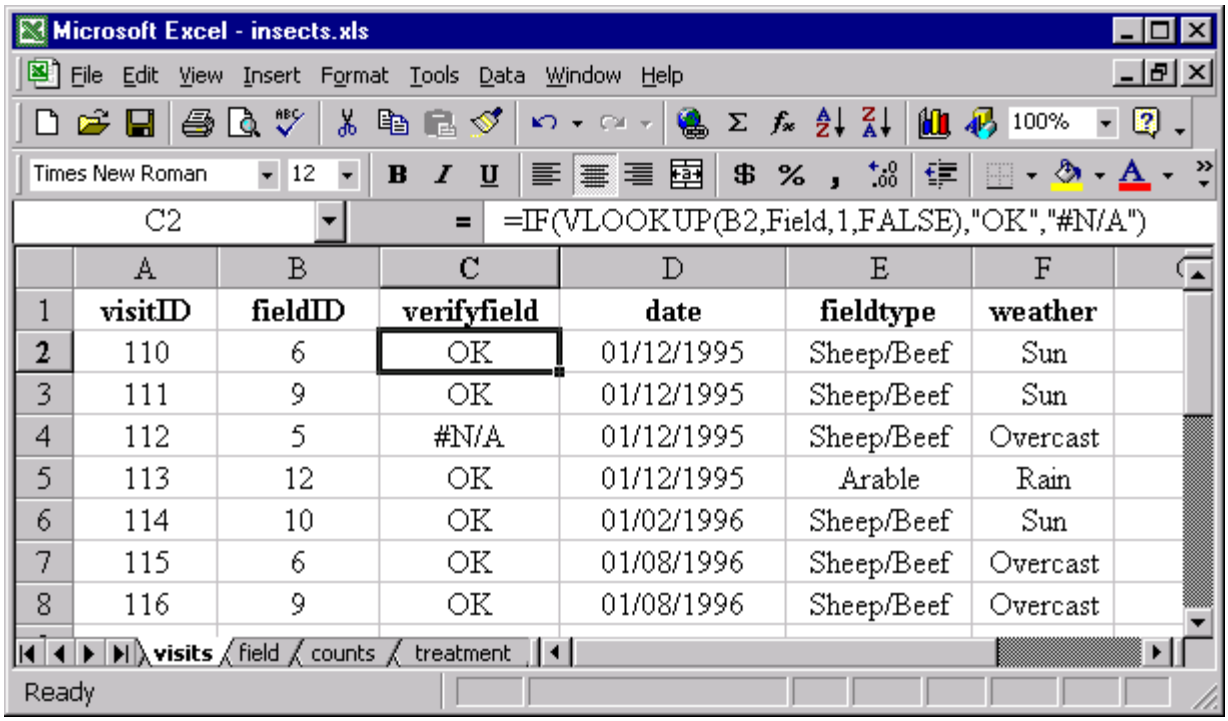
ii. Use exact match LOOKUPS for the relationships

iii. Set up validations for the constraints

iv. Set up one sheet combining information from the others to act as a view.

We discuss each of these in turn

i. Construct properly set up spreadsheet lists on separate sheets for each class. This enables rows to be added and deleted to one set of data (visits say) independently of the other sheets. In Figure 4 you can see that 5 sheets have been set up, one for each class.



	A	B	C	D	E	F
1	**visitID**	**fieldID**	**verifyfield**	**date**	**fieldtype**	**weather**
2	110	6	OK	01/12/1995	Sheep/Beef	Sun
3	111	9	OK	01/12/1995	Sheep/Beef	Sun
4	112	5	#N/A	01/12/1995	Sheep/Beef	Overcast
5	113	12	OK	01/12/1995	Arable	Rain
6	114	10	OK	01/02/1996	Sheep/Beef	Sun
7	115	6	OK	01/08/1996	Sheep/Beef	Overcast
8	116	9	OK	01/08/1996	Sheep/Beef	Overcast

Figure 4: Using a vlookup to represent referential integrity between sheets

ii. Use exact match LOOKUPs to simulate referential integrity between sheets and to verify that, for example, the fieldID in the visit sheet exists in the field sheet. In Figure 4, column C (verifyfield) is calculated using a lookup to the field sheet where all the information about each field is kept. We can immediately see that the field with ID 5 does not exist on that sheet. This gives some measure of referential integrity although there is no facility for cascade updates and deletes.

iii. Set up validations on columns to capture the constraints. In Figure 5 we see that the constraint on the value of weather has been implemented with a data validation list. This enables the user to pick values off a drop down list and prevents other values being entered. fieldtype has been validated similarly.

iv. Set up default input forms. In Figure 6 we show an example based on the counts sheet. On the counts sheet we have included calculated fields which combine information from the other tables. This allows the user to enter the information about counts and to get immediate feedback as to whether values such as visitID are valid. One problem in using Excel 2000 for this, is that it does not ensure that a validation on a column is respected when the data is entered through a form. However this can subsequently be checked using the auditing tools.

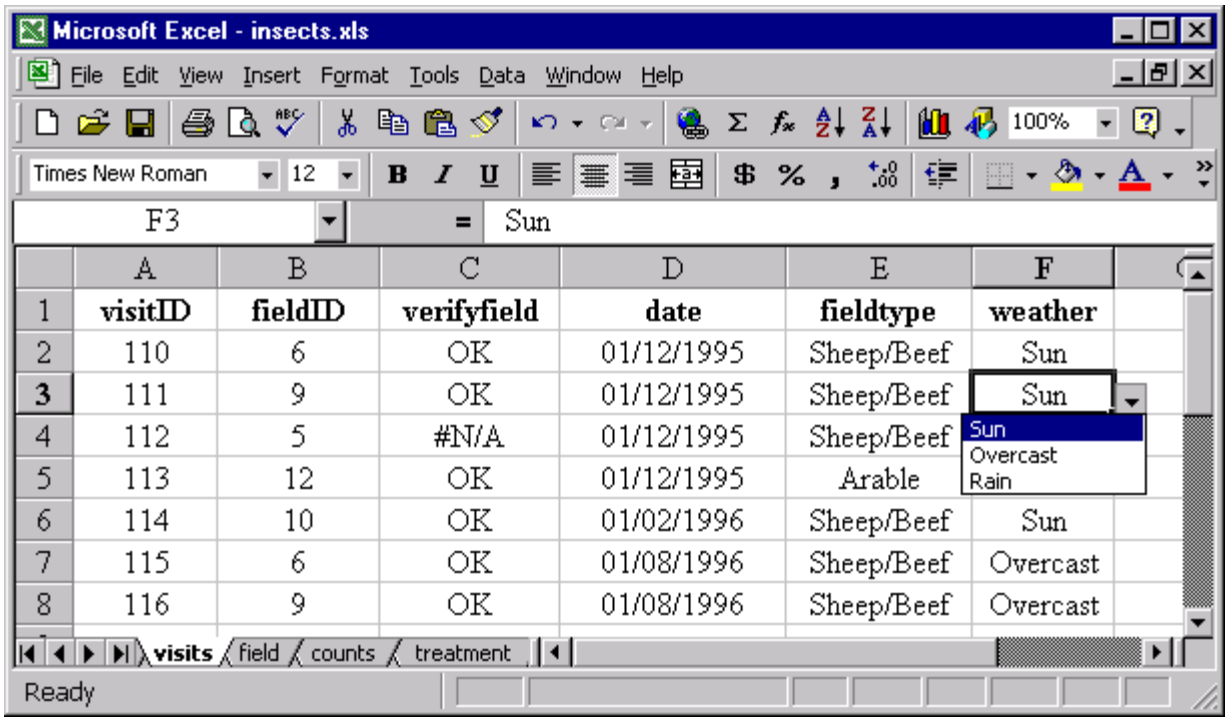


Figure 5: Using data validation to represent a constraint

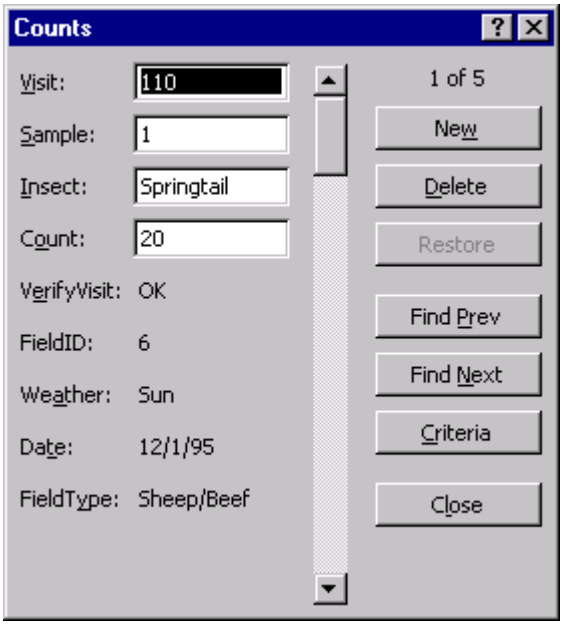


Figure 6: Input form for entering insect counts

v. Combine information from several sheets onto a single sheet (as we have done on the counts sheet with lookups such as those in Figure 4). This is analogous to creating a view in the relational model and helps overcome the limitations of many spreadsheets that make querying across multiple sheets difficult.

There are repercussions of the design in Figures 4 and 5. The user is much more responsible for the validation of the data input and complex queries (equivalent to a self join for example) are impractical. However the advantages over a single flat sheet are that redundant and therefore inconsistent data can be avoided. It can also be more easily exported to a well designed database. A database is clearly a more suitable tool for this type of data, but users who do not have the appropriate skills, confidence or access to a relational database can, after receiving help with the design, set up a useful data repository in a spreadsheet.

7. Extending the system

One of the advantages of creating a full conceptual model early in the design process is that there is a clear path for extending the system as resources become available. The first way is to gradually remove some of the simplifications if this is thought to be sensible. It is not too difficult a job to replace the constraints with new tables and Foreign Keys. Classes left out of the full model can be added as new tables without too much affect on the existing data. If the problem itself becomes larger requiring additional classes then the system is well placed to cope with these being added. The new classes can be added to the conceptual model and the only adjustments will be to those classes with which they interact.

If the implementation has been carried out in a spreadsheet as described in Section 6 then it is readily upgraded to a database if that becomes practicable. Each sheet can be exported to a table in the database and then the referential integrity and other constraints added.

The steps for extending a simplified spreadsheet implementation are:

1. Export spreadsheet to a database with each sheet becoming a separate table
2. Replace constraints by references to new tables as richer data is required
3. Add new classes/tables as the problems evolves.

8. Conclusion

End users often develop a data repository which satisfies their immediate needs only. They generally do not anticipate the problems that will arise from the lack of design and planning. In this paper we have discussed an example of a typical scientific spreadsheet. We explain why developing a full conceptual model helps the end user to understand the subtleties of their data, allowing them to make an informed decision as to their most important and immediate requirements. We then suggest a number of ways that the model can be simplified and how this can then be implemented in either a database or a spreadsheet. We also provide a route for extending the model as time, resources and skills become available. We provide checklists of steps for the simplification, implementation and extension phases

The rationale behind this approach is to allow the end user to develop an application that is both accurate and manageable. A simple but poorly designed end user application will suffer from problems which will affect the usability and accuracy of the system. At the other extreme, a typical end user will not have sufficient skills to implement and use an overly complex design.

We aim for a middle ground where the most important requirements can be captured accurately and as simply as possible. In this way the user can be confident of the integrity of their system and still retain the immediacy and control that is so valuable.

The example we have presented is typical of scientific or research data. The general principles are equally applicable in a commercial environment. Although payroll and general accounting data are probably best dealt with by generic off the shelf packages, many organizations may have other more specific requirements which they may attempt to meet by developing an end user application. Clubs and societies often keep information about their members and activities in spreadsheets and small businesses may want to track the progress of jobs through some specialized processes. Adapting complex packages to meet very specific tasks is not a viable option for many users. As with the scientific example, getting help to develop a complete conceptual model will help the user choose the best way to proceed.

9. References

Adamson, C . and Venerable, M. (1998). Data Warehouse Design Solutions. John Wiley and Sons.

Booch, G. and Rumbaugh, J. (1995). Unified Method for Object-Oriented Development. Rational Software Corporation.

Chen, P. P. (1976). The entity-relationship model – toward a unified view of data. ACM Transactions on Database Systems, Vol. 1, No.1, pp 9-36.

Churcher, C., McLennan, T. and McKinnon, A. (2000). Pragmatic Data Modelling and Design for End Users. 7th Asia-Pacific Software Engineering Conference Singapore. pp120 – 126.

Churcher, C. and McNaughton, P. (1998). There are bugs in our spreadsheet: Designing a database for scientific data. Research Report No 98/02 Centre for Computing and Biometrics, Lincoln University http://www.lincoln.ac.nz/amac/publish/acms/9802abst.htm.

Delligatta, A. (1992). Managing End User Computing: Empowering the User Community. A Guide for IS Managers." Information Systems Management, Summer, pp 63-64.

de Marco ,T. (1979). Structured Analysis and System specification. Englewood Cliffs,N.J. Prentice Hall.

Fowler, M. (1997). Analysis Patterns: Reusable Object Models. Addison-Wesley.

Gunton, T. (1988). End User Focus. Prentice Hall.

Halloran, J. P. (1993) Redefining End-User Computing: Achieving World-Class End User Computing. Making IT Work and Using IT Effectively." Information Systems Management, Fall, pp 7-12.

Hay, D. (1996). Data Model Patterns: Conventions of Thought. New York, Dorset House.

Hobbs, V. and Pigott, D. (2000). Facilitating end user database development by working with users' natural representation of data. Research Working Paper , Information Technology, Murdoch University IT/00/03.

Jackson, M.A. (1975). Principles of Program Design, London, Academic Press.

Kimball, R., Reeves,L., Ross, M., and Thornthwaite, W. (1998). The Data Warehouse Lifecycle Toolkit. John Wiley and Sons.

Kreie, J., Cronan, T., Pendley, J. and Renwick, S. (2000). Applications developments by end-users: can quality be improved? Decision Support Systems 29 pp143-152.

Lincoln University Computer Industry Liaison Group minutes (1999) (unpublished).

Martin, J. and Odell, J. (1998). Object-Oriented methods: A Foundation. Prentice Hall.

McLennan, T., Churcher C. and Clemes S. (1998). Should End User Computing be in the Computing Curriculum? pp 346-352, Proceedings of SEE&P, Dunedin, New Zealand.
Panko, R. R. (1987). Directions and Issues in End User Computing. INFOR, vol 5, no. 3, pp 181-197.
Panko, R. R. http://panko.cba.hawaii.edu/ssr/ Cited 2001.
Shlaer, S., and Mellor, S. (1988). Object-Oriented Systems Analysis: Modelling the World in Data. Prentice Hall.
Warnier, J.D. (1977). Logical Construction of Programs. New York Van Nostrand.
Yourdon, E. (1989). Modern Structured Analysis. Englewood Cliffs, N.J. Prentice Hall.