

Pragmatic Data Modelling and Design for End Users.

Clare Churcher, Theresa McLennan and Alan McKinnon
Applied Computing, Mathematics and Statistics Group
Lincoln University
P.O. Box84, Lincoln University
Canterbury
New Zealand

email:churchec/mclennan/mckinnon@lincoln.ac.nz

Abstract

Many people are dependent on desktop end user tools such as spreadsheets and databases to manage their data. While they may have the technical skills to set up data repositories, many end users lack the analysis skills to design data models which reflect their often deceptively complex requirements. We advocate that a comprehensive data model should always be developed, with expert help, so that the end user can feel confident the subtleties of the data are fully understood. We then suggest that some pragmatic decisions can be made to simplify the model so that the end user can retain control over setting up and maintaining the application.

1. Introduction

Data modelling has evolved considerably since the early entity-relationship models [1]. A number of methodologies and notations have been proposed as the transition has been made to more object oriented models [2-4]. There has also been considerable work in identifying patterns of data models and in applying these to a variety of problems [5, 6].

Nevertheless much data is still kept in poorly designed systems. The news media regularly have stories of costly and damaging failures of large database projects. We do not wish to address the causes of these large failures, however the difficulty in successfully implementing corporate wide systems often results in frustrated end users developing numerous small systems themselves. These applications are seldom carefully designed and problems inevitably arise: data becomes inconsistent, queries can't be answered, statistics and analyses have errors. These types of problems occur in many databases and are well documented for spreadsheets [7].

A precise understanding of the data is critical to making any decision about how to store it. Most data is

deceptively complex and producing a comprehensive and accurate data model is a valuable exercise to gain insight and understanding. We would encourage any keeper of data to seek expert advice in producing a comprehensive data model for their problem.

Many end users have neither the time, the resources nor the skill to fully and faithfully implement a complex data model. We advocate taking the most critical features of the comprehensive data model and producing a simpler design for a database or even, in some cases, a spreadsheet. This relies on a very clear understanding of what the user requires as the most important outputs of the system; specific reports, statistics and so on. Some of the less important constraints on the data may be removed from the design and the responsibility placed at the interface or even with the user of the application. This needs to be done with the user fully aware of the compromises he or she is making and how they will affect the output. We advocate that a simple (but accurate) system can be developed which can grow in a controlled way to include extra features as time and resources become available.

2. Who are the end users?

More and more users are now 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 [8 – 10].

There is a considerable variety of people who 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 [11]. The end user focus is typically narrow and immediate.

This is 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 most immediate analytical needs [12]. 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. Strategies for creating end user applications

A common scenario for an end user application is that the user takes full control of the design, implementation and maintenance of the system as in Figure 1a. At the other extreme an expert can be employed to take over the whole job as in Figure 1c. This scenario is unlikely in that

the major benefits of end user tools, immediacy, cheapness and control, are likely to be lost.

While many end users have the ability to undertake the mechanics of setting up a spreadsheet or database with desktop tools, analysis and design skills are much scarcer. The scenario in Figure 1b shows that an expert can be used to work with the end user to gain a complete understanding of the data, and then produce a simplified model that is within the user's capabilities to implement. In this way the end user has a much greater understanding of the problem, is aware of the constraints of the system under construction and, most importantly, retains control of the system.

The cost, in both time and money, of employing an expert must be weighed against the problems likely to be incurred with a poorly designed data repository which may produce inaccurate results and be difficult to maintain.

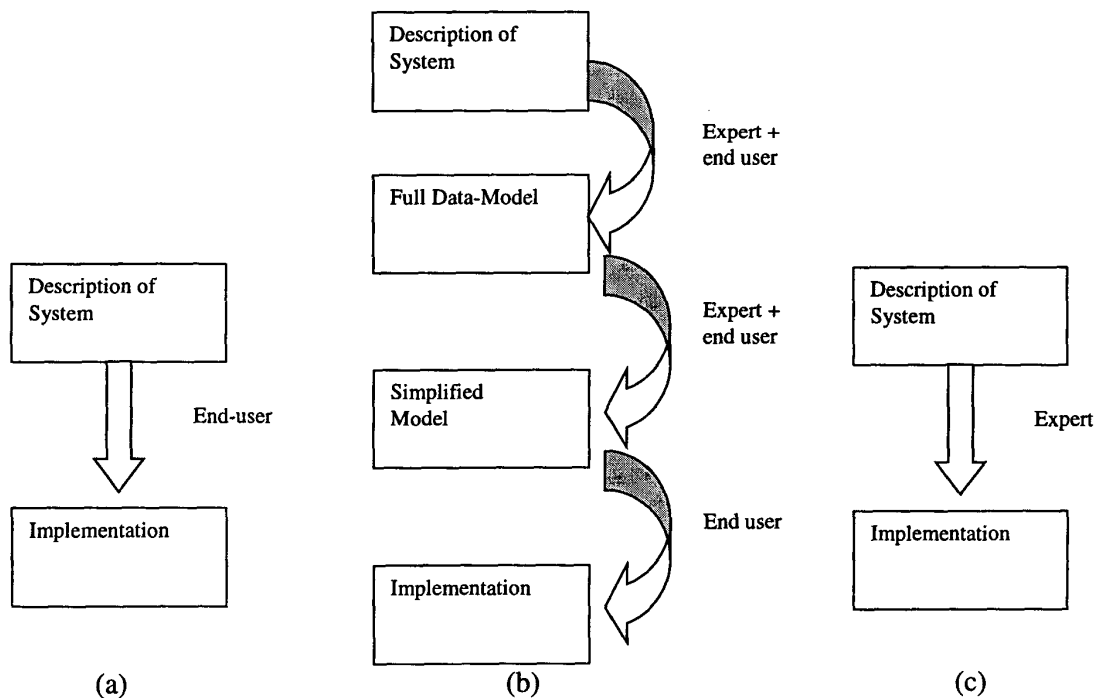


Figure 1. Possible strategies for creating a database repository with end user tools

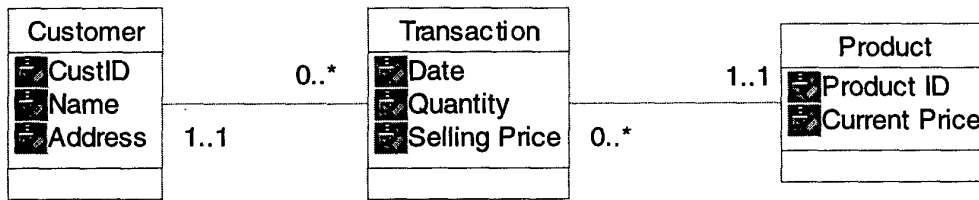


Figure 2. A simple data model

4. A simple example

Most small businesses require a repository for data about their customers and products or services. The customer – order – product model is a standard text book example. Orders are for a product (possibly many) and each order is related to a customer. It can be extended to cope with backorders and so on (but most elementary texts don't bother).

A variant of this problem could be adopted by a small business wanting to invoice customers for credit purchases and keep track of the number of products it sells. Rather than orders we can consider transactions and obtain a data model as in Figure 2. To keep it simple we will define a transaction as being for only one type of product.

The real world is never quite so simple and a good data analyst will ask "what about cash sales for which (by definition) you don't want to record the customer?"

There are a number of possibilities.

- remove the compulsory requirement at the customer end of the relationship
- have a customer object called cash customer.
- have different specialisations of transaction.

The most comprehensive solution is the last one, which accurately reflects the fact that there are two types of transaction: an account transaction which must have a customer associated with it for invoicing purposes, and a cash sale that by definition does not have any associated customer details. This model, which can be arrived at together by the end user and an expert data modeller, is shown in Figure 3.

Users will, quite reasonably, argue that the other solutions "will work" and "are easier". It is unlikely that any end user would actually implement anything as complex as Figure 3 for what is only a small part of the system (we haven't yet included payments, or stock on hand, or deliveries).

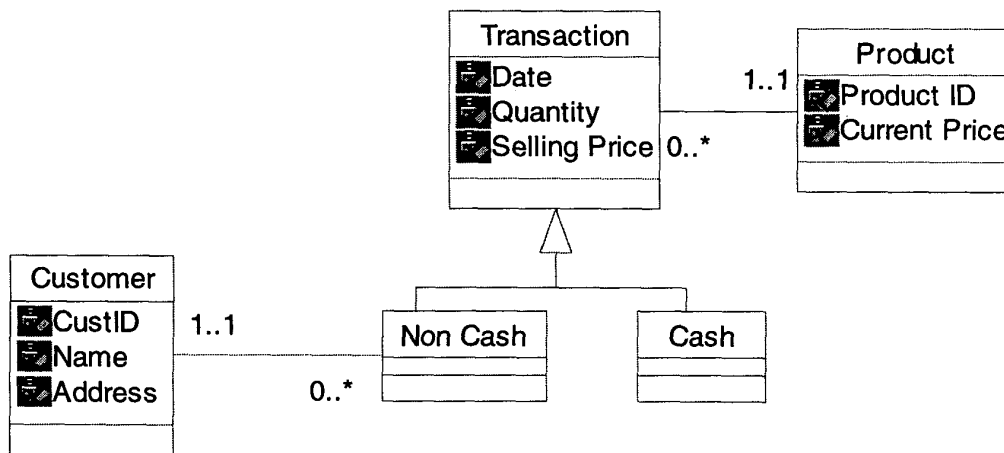


Figure 3. A comprehensive data model

So why bother to construct the comprehensive model at all?

The comprehensive model is useful because it describes the situation fully (well at least as fully as we have described it here). If this model is implemented we can be confident of doing all of the following:

- Ensure all non-cash transactions have customer details included
- Produce invoices for non-cash transactions
- Report the number of each product sold grouped by date (for seeing trends)
- Report the income from different products
- Provide statistics about how much of each product a typical non-cash customer buys in a year

With an implementation following this model it will also be possible to make the following types of extensions to the transactions with relatively little impact on the existing data.

- Record if cash sales are by cheque or EFTPOS or cash or card (and maintain any extra information that might be needed for some of these types of sales)
- Include a subset of non-cash sales which might be paid in instalments (time payment etc)

5. Solutions and repercussions

5.1. Schema representing the comprehensive data model

In an Object Oriented database the model in Figure 3 can be represented faithfully. Each transaction would be represented by one object of either the cash or noncash

Product	<u>[ProductID]</u> , <i>product info: price,.....</i>]
Customer	<u>[CustomerID]</u> , <i>customer info: address, ...</i>]
Transaction	<u>[TranID]</u> , (ProductID), Quantity, Date, ...]
CashTransaction	(<u>[TranID]</u>), <i>chequeno, ...</i>]
NonCashTransaction	(<u>[TranID]</u>), (CustomerID), <i>interest rate,....</i>]

(primary keys are underlined, foreign keys are in (brackets))

Figure 4. Relational representation of the comprehensive data model

Product	<u>[ProductID]</u> , <i>product info: price etc,</i>]
Customer	<u>[CustomerID]</u> , <i>customer info: address etc</i>]
Transaction	<u>[TranID]</u> , (ProductID), (CustomerID), Quantity, Date, ...]

Figure 5. Simplified relational schema

specialisations. Even in a relational database we can capture most of the features of the model although some of the responsibility for correct maintenance shifts from the database software to the developer of the application software. A possible relational schema is shown in Figure 4.

Every transaction will have a record in the transaction table and also one in either the cash or noncash transaction tables (with the TranID referencing the transaction table). Having two records for each transaction is not as “nice” as having one object in an OO schema, but a good user interface can help to make it transparent to the user. The developer has to take the responsibility to join the three transaction tables appropriately. Unlike the OO implementation this relational design allows for transactions to be both cash and non cash. This multiple inheritance type behaviour may or may not be useful depending on the type of problem. In either case it is the developer’s responsibility to ensure it is handled correctly.

If it is an essential aspect of the system to maintain different information about cash and noncash transactions then a model such as the one in Figure 3 is necessary. This model will also allow for extra classes to be added should the system need to be extended to recording a series of payments for each transaction. In the relational schema in Figure 4 this could be handled quite readily by the addition of another relation.

Payments [(TranID), date, amount,].

The foreign key would reference the NonCashTransaction table and the developer would have to ensure that all the joins were correctly implemented.

5.2. Simplified schema

If the problem does not require the system to maintain data or behaviour specific to the different types of transaction, we can simplify the schema considerably by including a foreign key CustomerID for all transaction records as in Figure 5. However the developer or user will have to take on board some extra responsibilities.

It is possible to distinguish a cash or non cash payment using the schema in Figure 5 in two ways.

We can leave the CustomerID field, in the transactions table, empty for cash sales. The problem in this case is that a null in the CustomerID field could indicate either that there is no customer (ie a cash transaction) or that it is a non cash transaction for which the customer is unknown or whose details have not yet been recorded. The responsibility has now shifted to the user to make sure that all non cash transactions have an associated customer. The analyst should ask: "Is it crucial that the system determines which transactions must be billed?" If yes then having a null default implying a cash sale is an unsatisfactory solution. For a small business however, the user may be quite happy to be responsible for checking which transactions are non cash and so require billing.

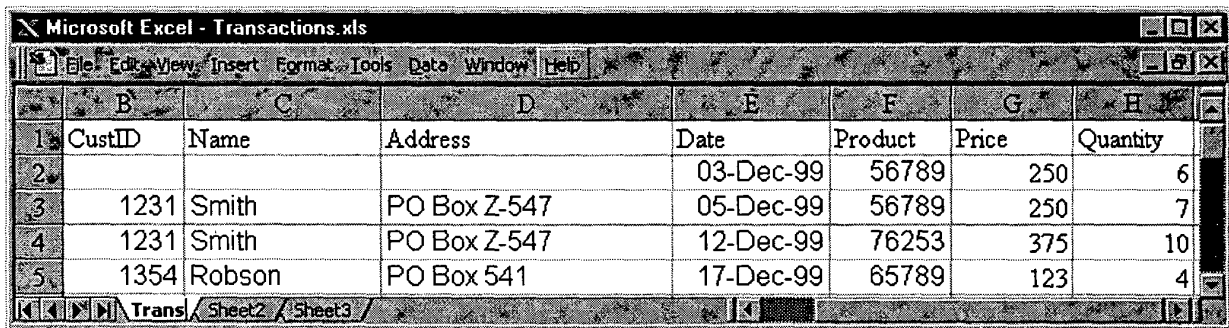
Another way to identify cash sales is to have an entry in the Customer Table called "Cash" or similar. The user now has to consciously specify that the transaction is a cash transaction rather than that being assumed if the field is left blank. There are a number of potential problems. The "Cash" customer record will have to have a CustomerID number. This needs to be chosen carefully so that it doesn't eventually turn into a "Customer 2000 Bug". All the reporting and analysis facilities will have to take this dummy customer into

account. Reports on number of customers or average sales for a customer and so on will all be distorted by the dummy record. While it is possible to work around these problems it is ugly and potentially dangerous. The analyst should ask: "Is analysing statistics about your customers' transactions a crucial part of this system". If the statistics are important then the dummy record will be an unsatisfactory solution.

5.3. Representing the model in a spreadsheet

End users are more likely to feel they are competent users of spreadsheets than they are of databases [13]. Certainly, until recently users were more likely to have access to spreadsheet than database software. Even now only the professional versions of the popular office suites (eg Microsoft Office and WordPerfect Office) include a relational database. Informal discussions with user support managers [14] suggest that it is not uncommon for end users in a large organisation to be denied access to a relational database 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 not practical to try to capture the complexities of a comprehensive data model such as Figure 3 in a spreadsheet. However for a simplified schema such as in Figure 5 we can capture many of the features. Certainly there are improvements which can be made over the simple flat design with all the information on a single sheet as depicted in Figure 6. This design has significant problems with information being stored several times with the potential to become inconsistent (e.g. a customer's name and address).



	B	C	D	E	F	G	H
1	CustID	Name	Address	Date	Product	Price	Quantity
2				03-Dec-99	56789	250	6
3	1231	Smith	PO Box Z-547	05-Dec-99	56789	250	7
4	1231	Smith	PO Box Z-547	12-Dec-99	76253	375	10
5	1354	Robson	PO Box 541	17-Dec-99	65789	123	4

Figure 6. A flat spreadsheet design

Microsoft Excel - Diveshop.xls

File Edit View Insert Format Tools Data Window Help

C3 =IF(VLOOKUP(B3,allCust,1,FALSE)<>B3,"#N/A","ok")

	A	B	C	D	E	F	G
1	TranID	CustomerID	Verify Customer	Customer Name	Date	Product ID	
2	1001	1221	#N/A	#N/A	03-Dec-99	56789	
3	1002	1231	ok	Smith	05-Dec-99	45635	
4	1003	1231	ok	Smith	12-Dec-99	76253	
5	1004	1354	ok	Robson	17-Dec-99	56737	
6	1005	1356	ok	Johnson	20-Dec-99	23444	

TRANS CUSTOMER

Figure 7. Simulating referential integrity with a Lookup in Excel

Some of the techniques that can be used to represent a schema such as that in Figure 5 are described below and shown in Figure 7.

- Use separate sheets for each table. This enables rows to be added and deleted to one set of data (customer say) with the least disruption to any other information (transactions or products). It also simplifies later extraction to a database when resources become available.
- Use exact match LOOKUPS to simulate referential integrity between sheets and to verify that, for example, the customer number in the transaction sheet exists in the customer sheet. In Figure 7 both columns C and D are calculated using lookups to the customer sheet. If a customer is deleted from the customer sheet the rows in the transaction sheet for that customer will reflect the change. This gives some measure of referential integrity although there is no facility for cascade updates and deletes. Tools such as Data Validation in Excel would be another way of achieving a similar result.
- Set up default input forms such as in Figure 8 which give feedback as to whether a

CustomerID is valid and also prevents any tampering with the calculated fields. A complication is that in most common spreadsheets, forms are, by default, only for one sheet at a time.

- Combine information from several sheets onto a single sheet (by using lookups as in Figure 7). This is analogous to performing a join in the relational model and helps overcome the limitations of many spreadsheets that make querying across multiple sheets difficult.

The repercussions of the design in Figure 7 are that 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 and that it can 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.

TRANS

TranID: 1002

CustomerID: 1231

Verify Customer: ok

Customer Name: Smith

Date: 05/12/1999

Product ID: 45635

2 of 224

New

Delete

Restore

Find Prev

Find Next

Figure 8. Input form for a transaction

6. Discussion

A good understanding of the data is critical if any data repository is going to yield accurate results. An expert data modeller can provide the skills to produce a comprehensive data model. If, as is usually the case for an end user, the necessity to develop and keep control of their small applications is important, then a simplified model can be constructed which captures the crucial aspects of the system accurately. This means that some responsibility for maintaining and manipulating the data accurately may be shifted to the interface or the user. Once again an expert data modeller can determine the repercussions of the simplifications and alert the end user to how they can be most effectively managed.

The pragmatic data model will produce results whose accuracy is well understood. More importantly, the crucial aspects of the data will be maintained correctly so that subsequent extension to a more complete model can be carried out.

7. References

- [1] Chen, P. P. The entity-relationship model – toward a unified view of data. *ACM Transactions on Database Systems*, Vol. 1, No.1, pp 9-36. 1976.
- [2] Shlaer, S., and Mellor, S. *Object-Oriented Systems Analysis: Modelling the World in Data*. Prentice Hall, 1988.
- [3] Martin, J., and Odell, J. *Object-Oriented methods: A Foundation*. Prentice Hall, 1998.
- [4] Booch, G., and Rumbaugh, J. *Unified Method for Object-Oriented Development*. Rational Software Corporation, 1995.
- [5] Hay, D. *Data Model Patterns: Conventions of Thought*. New York, Dorset House, 1996.
- [6] Fowler, M. *Analysis Patterns: Resusable Object Models*. Addison-Wesley, 1997.
- [7] Panko, R. R. <http://panko.cba.hawaii.edu/ssr/>
- [8] Panko, R. R. "Directions and Issues in End User Computing." *INFOR*, vol 5, no. 3, 1987, pp 181-197.
- [9] Delligatta, A. "Managing End User Computing: Empowering the User Community. A Guide for IS Managers." *Information Systems Management*, Summer 1992, pp 63-64.
- [10] Halloran, J. P. "Redefining End-User Computing: Achieving World-Class End User Computing. Making IT Work and Using IT Effectively." *Information Systems Management*, Fall 1993, pp 7-12.
- [11] Gunton, T. *End User Focus*. Prentice Hall, 1988.
- [12] Churcher, C. and McNaughton, P. There are bugs in our spreadsheet: Designing a database for scientific data. Research Report No 98/02 Centre for Computing and Biometrics, Lincoln University 1998.
http://www.lincoln.ac.nz/amac/publish/acms/9802a_bst.htm
- [13] McLennan, T., Churcher C. and Clemes S. Should End User Computing be in the Computing Curriculum? pp 346-352, *Proceedings of SEE&P*, Dunedin, New Zealand 1998.
- [14] Minutes of the Lincoln University Computer Industry Liaison Group 1999 (unpublished).