Data Analysis and System Design by Entity-Relationship Modelling
A Practical Example

A. Parkin
Leicester Polytechnic, PO Box 143, Leicester LE1 9BH, UK

A published case study from the 1960s is dis-analysed and the assumptions and limitations revealed. The case is redesigned and the new solution is contrasted with the original.

INTRODUCTION

Entity-relationship modelling[1] is valuable because it gives systems analysts a language for thinking in. The aim of this paper is to illustrate the thought processes inspired by the technique, by taking a practical example which contains a realistic level of complexity. The reader is forewarned that to follow the fine points of the discussion he will have to make a substantial investment of his time. Readers who are unfamiliar with the principles of system design through data analysis should read Ref. 2.

A technical difficulty in systems analysis is the conception of an efficient overall system design, complete enough to fulfil all present requirements and robust enough to survive expansion and change. This difficulty is severe when the requirements are complex. The strength of the entity-relationship model lies in its ability to cut through the complexity, helping the analyst to explore alternative high-level decisions. A further value is the aid it gives to raising questions which should be put to management prior to concluding the design.

An expository difficulty in explaining the use of the technique with a realistic example is the need to provide a rich set of facts describing the case, facts which the expositor should avoid marshalling in too orderly a fashion lest he destroys the point of the example. The facts which practising analysts acquire for background are usually discovered over a period of time, and arise in a disorderly fashion which is hard to simulate. This paper compromises with background facts from an old case whose design was not influenced by data analysis thinking. This also allows the process of dis-analysis to be illustrated. Dis-analysis is the construction of a data model from the record layouts of master and transaction files in a system. This is useful for interpreting system specifications for purposes such as tuning, estimation, maintenance and package evaluation.

Presumably the analysts concerned with the original design faced an even richer set of facts than those recorded. I played no part in the case and have no special knowledge of the type of business, a bakery. Therefore I share with the reader the need to interpret the case purely at face value; the need to question the facts and to speculate about points of detail. The mention of limitations of the original design is certainly not meant as any slur on the analysts concerned, whose priorities and design philosophy must be judged in the context of the time. The reader should now study the facts in case study two, Order Control and Sales Accounting, Ref. 3.

DISCUSSION OF BACKGROUND

Figure 1 is my attempt to describe the feature facts on a single page: the principal inputs, outputs and reports; the overall data flow; the principal processes and their frequency; the main master and transaction files needed to support each process; which processes are supported by each file. It is usually safe to omit from high-level consideration workfiles, recovery files and recovery procedures, control and error reports and other routine procedures which are only secondary to the productive processes leading to the principally desired results. No doubt though there are exceptional cases where the tail must wag the dog. Figure 1, by omission, judges the temporarily stored commodity totals in run B22 to be a workfile, but this is debatable.

In section 2.3 of the case, it says that the sales analysis information in the commodity master and analysis file covered the previous 12 weeks. Later it appears to cover 8 weeks and 24 months; the latter is assumed to be correct.

In section 2.4 of the case, the use of the standing/required/delivered quantities by runs B20, B21 and B22 (section 2.5) is rather difficult to follow. There is more than one way in which these fields could be physically used to achieve the desired result, which is that if there is no return or adjustment, the required quantity is deemed to have been delivered, and if there is no order amendment, the standing quantity is deemed to be the required quantity. One interpretation is that for each day (Monday, Tuesday, ...) there are three quantities—a standing quantity, a required quantity and a delivered quantity (see Fig. 2). When a delivery note is produced for a day, the required quantity is moved to the delivered quantity for that day, and the standing quantity is moved to the required quantity, which now stands for the required quantity on that day next week. Just after the delivery notes are produced, the six required quantities for a given commodity make a (circular) list of the required quantities for the next six days, starting the day after the day to which the delivery notes referred. At the weekend, the six delivered quantities will represent the quantities delivered over the preceding six days, as amended by returns and adjustments. These six quantities

CCC-0010-4620/82/0025-0401 $04.50

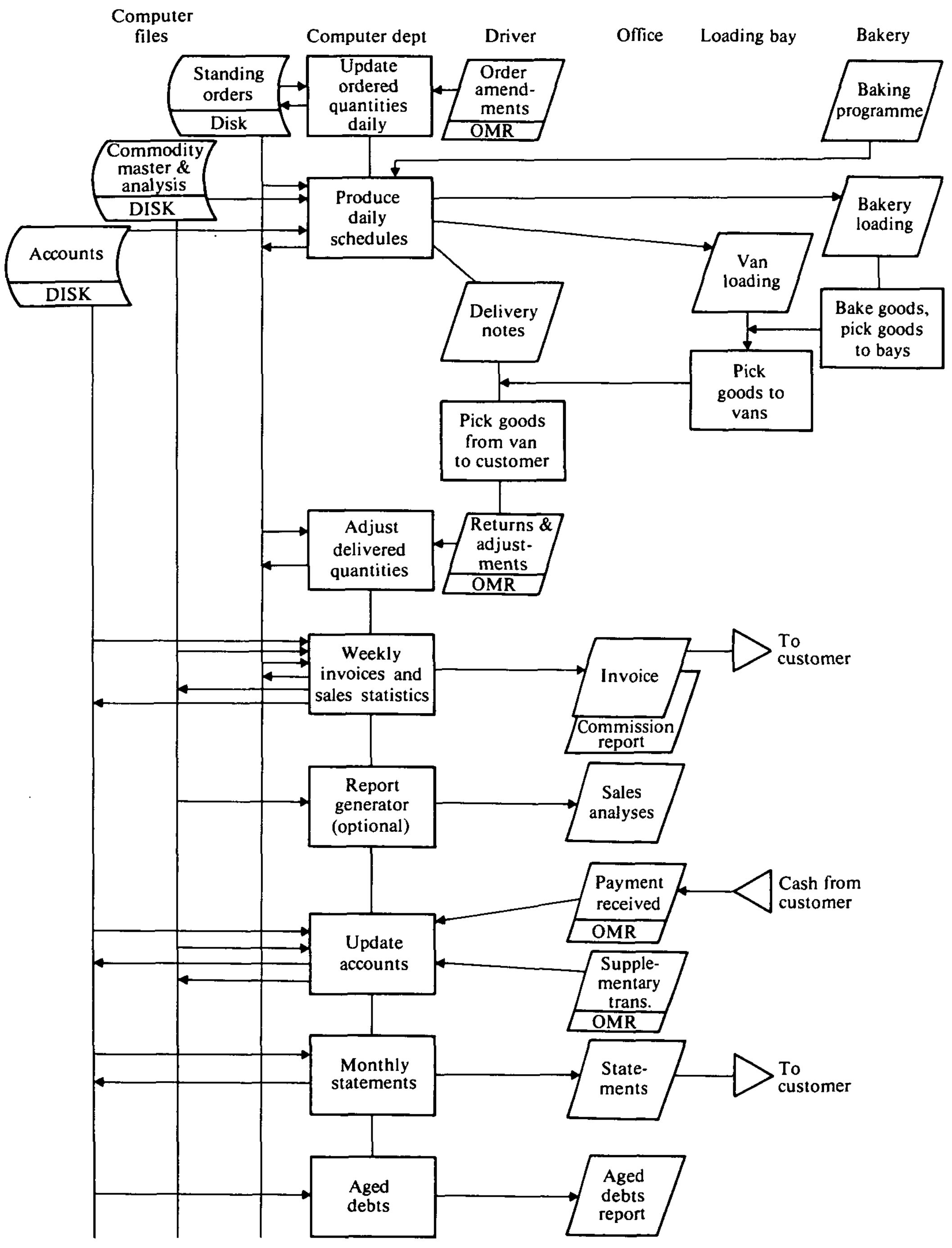


Figure 1. The overall system.

Customer 027, 1lb sliced white

	Standing quantities						*Required quantities*						*Delivered quantities*					
	M	T	W	Th	F	S	M	T	W	Th	F	S	M	T	W	Th	F	S
(a)	100	100	100	100	100	200	50	100	100	100	200	200	100	100	100	100	0	0
(b)	100	100	100	100	100	200	50	100	100	100	100	200	100	100	100	100	200	0
(c)	100	100	100	100	100	200	50	100	100	100	100	200	0	0	0	0	0	0

Figure 2. A possible interpretation of the use of the 18 customer/commodity quantities. Time (a), before production of Friday's delivery notes; Time (b), just after production of Friday's delivery notes; Time (c), after the invoice has been prepared.

are then set to zero after they have been accounted for in the weekly invoice. Figure 2 illustrates this idea. At time (a), before production of Friday's delivery notes, order amendments have been received for this Friday and next Monday. Last Monday's order was not amended. At time (b), just after producing Friday's delivery notes, the required quantity is assumed to be safely delivered and the required quantity next week is assumed to be back to normal. At time (c), when the invoice is prepared between Saturday and Monday delivery note production, the recorded delivered quantities are used to bill the customer and then reset to zero. Let us assume that this is a legitimate interpretation of the use of these fields.

Returns and adjustments which arise after preparation of the weekly invoice, but which relate to a delivery prior to the invoice, are not offered to the same process since there would be wrong amounts allowed if a price change occurred between the original delivery and the invoice preparation. Also, customers would be unable to reconcile the deliveries claimed in the invoice. These returns and adjustments, not processed in the referent week, are one source of supplementary transactions in run B25 (presumably of class 3 or class 4). The frequency of processing these, and the payments received etc., is not stated. It should be at least monthly and is presumably weekly or daily.

The home-made index for the accounts file (section 2.4) seems to introduce an awkward limitation if one of the multiples decides to increase its number of shops in the area; presumably the storage is allocated so that there is a slot for at least one extra branch in each multiple.

Another interesting feature is the sales history on the commodity record, which is supposed to cover a two-year period broken into 24 months, the last two months being further divided into eight weeks. A difficulty arises in interpreting this, because in two calendar months there may be nine weekly invoices raised. In run B23, it seems that the week's invoice amount just raised is inserted in the most recent week's figure and added to the most recent month's figure. This suggests that the months are calendar, but the corollary is that some monthly totals contain five weeks' invoices, and some four, according to where the Saturdays lie. The alternative interpretation is that the months are lunar, but then the totals would not span a two-year period unless the eight weeks were end on to the 24 months. Let us assume that the totals refer to calendar months and that sales comparisons are to be made on this basis in spite of the 25% perturbation.

The money values in the sales analysis reports do not seem to be stored anywhere, so presumably they are derived by using the current prices. This will mean that the past figures may vary with current prices; this could be interpreted as reporting the past in present value equivalent.

In the case as implemented, an estimate of the file storage used for master and transaction files is:

Standing orders file (indexed)		
Data: 10 900 × 64	697 600	
Provision for 38% oflo	265 088	
Indexes: 27 × 40 × 10 + 270	11 070	
	973 758	973 758
Commodity master and analysis file (relative)		
Data and spare: 499 × 4456		2 223 544
Accounts file (special index)		
Data 292 × 719	209 948	
Spare or wasted: 292 × (1000 − 719) + (360 − 292) × 1000	150 052	
	360 000	360 000
Total characters		3 557 302

In a byte-oriented environment, it is sensible to get this global picture divided between alphanumeric characters and numeric digits, since packing the numbers may make for significant compression.

The explanation that follows will be more concise if file and attribute names are abbreviated:

SO, standing order; SQ, standing quantity; RQ, required quantity; DQ, delivered quantity.

COMM, commodity; C#, commodity code; PQ, quantity per pack; P£, pack price; PP, pack points; PD, pack description; WS, weekly sales quantity; MS, monthly sales quantity.

ACC, account; A#, account number; AC, account class; B%, bread discount; C%, confectionery discount; INA, invoice name and address; CNA, consignment name and address; CB, current balance of account and separate sign; PB, previous balance of account and separate sign; T#, transaction number, TC, transaction class; TD, transaction date; T£, transaction amount; TF, statement flag.

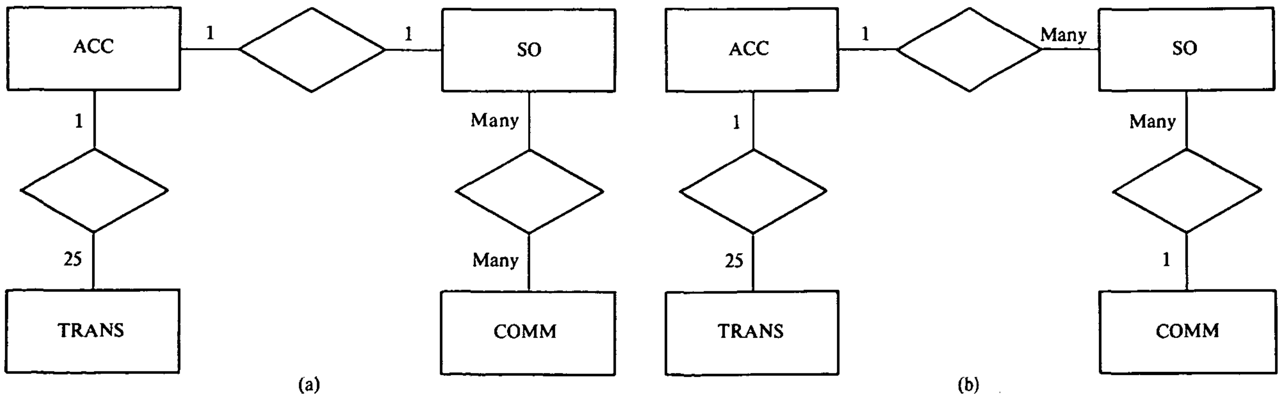


Figure 3. Alternative models (optionality of relationships neglected).

DIS-ANALYSIS OF THE EXISTING DESIGN

The aim is to derive a data model which is descriptive of the design implemented in the case. Clearly the accounts records are not fully normalized; the transaction (TRANS) group needs to be treated as a separate entity. Treating the other tabular entities as fully normalized (on the grounds that their table position allows the derivation of a unique name for each quantity), and for the moment accepting the idea of A# as a single attribute, the candidate entities/relationships are: account, commodity, standing-order and transaction. Two ways in which these are likely to be composed in an entity-relationship model are shown in Fig. 3.

The difference between these two is in the interpretation of the idea 'standing order'. Version (a) considers that a customer has a single standing order, covering all his desired commodities. This seems to be the vernacular use of 'standing order' in the bakery. In version (a), the SO–COMM relationship is concerned with the lines that appear in the standing order. In version (b), the meaning of standing order is the same as that of a record in the existing SO file, i.e. what is meant by SO–COMM in version (a).

These two models are of course essentially the same. The 1:1 ACC–SO relationship in version (a) can be simplified by merging those two entities. If we decompose the many:many SO–COMM relationship of version (a) into its 1:many, many:1 components, it is apparent that the alternative meaning of SO is revealed. This flexing is illustrated in Fig. 4.

There is no special key for SO, occurrences being identified by (A#, C#). The key of the relationship ACC–SO formed by pooling the keys of the two entities it joins will also be (A#, C#), as will be the key of the SO–COMM relationship. Since any attributes under this key can be stored with SO, the model could be simplified to that of Fig. 5.

As a general rule it is worthwhile to decompose many:many relationships, at least mentally, even if they are eventually documented in more compressed form.

The degree assumptions of Fig. 5, which should be checked out with user management, are as follows:

(1) A given transaction may concern only one account.
(2) A given account can have a maximum of 25 transactions.
(3) A given account may place standing orders for many commodities.
(4) A given commodity may be the subject of standing orders from many accounts.

The model of Fig. 5 corresponds closely to the records designed in the case, each entity and relationship being realized as a record type, the transaction record type being posted 25 times into the account record type. It is recognized in the case that this posting does not square with the facts, since accounts have a variable number of transactions in a two-month period. Also, unless there is some business rule or policy which will refuse the 26th transaction or assimilate it manually, the upper limit is presumably not reliable. The probability of an account having more than 25 transactions may be very small, but one could conjecture some systematic change in the future which would result in this limit being regularly exceeded. The design would be more robust if this limitation were avoided.

While on the subject of transactions, the flagged transactions seem to be a loose end since they are not used after they have appeared on the statement. Perhaps they are included in the aged debts report, or perhaps there is a plan to provide facilities to enquire about the current and previous month's transactions. In keeping with the general viewpoint of this paper, the required outputs and the facts that are to be stored may be questioned, but the discussion will continue on the basis of matching the facilities of the original system.

DEEPER ANALYSIS—A# FACETS, SPARE ATTRIBUTES

We now relax the assumption about A# being a single attribute, and recognize its facets, round number (R#),

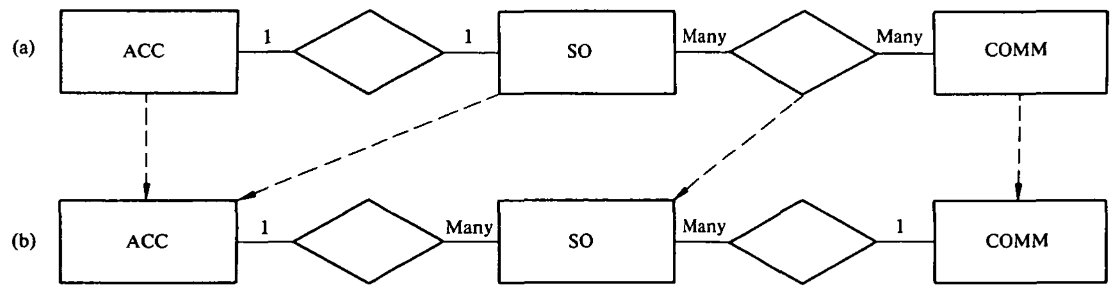
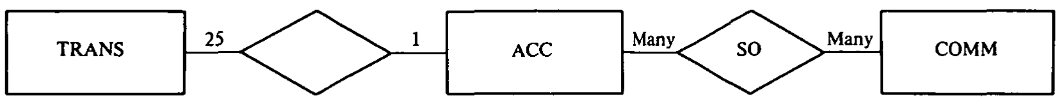


Figure 4. Alternative (a) can be flexed into alternative (b).



TRANS (T#, TC, TD, T£, TF)
TRANS-ACC (T#, A#)
ACC (A#, AC, B%, C%, INA, CNA, CB, PB(3))
SO (A#, C#, SQ(6), RQ(6), DQ(6))
COMM (C#, PQ, P£, PP, PD, WS(8,24), MS(24,24))

Figure 5. A simple representation of the existing design.

group code (G#) and branch code (B#). However, we still assume that these codes and C# are given, and not open to redesign at this stage.

We expect every data item of the case to be either an attribute of an entity or relationship of the data model, or a derivation of the former attributes, this derivation not needing to be stored, or a parameter of the processes of the system, being a data item which is introduced to a process but which does not need to be stored when the process is concluded. Attributes of the entities and relationships are eventually to be realized in the master and transaction files designed for the system. Derivations, such as totals and page counts, will be realized by processes, or by workfiles, work records or control records passed between processes. Parameters, such as today's date, will be realized in messages stored in or temporarily passed to the processes.

Working through the case, and neglecting data items such as indexes invented purely to support the chosen physical design, I classified every remaining data item mentioned as a derivation or parameter, apart from the commodity totals stored during delivery note production, and the loading bay letter (BAY#). I put a question mark over the commodity totals, but decided this was a point of detail which could be left out for the time being. BAY# could be treated as a parameter, but in view of its connection with R# it may be preferable to treat it as an item to be stored. This leaves it as a loose end in our present model. We cannot put BAY# in with entities which hold R# (e.g. ACC) since this would leave the entity not fully normalized. The pair (G#, B#) determines R#, and therefore BAY#, but R# alone determines BAY# whereas it does not alone determine other attributes of ACC. Separating out the entities ROUND and BAY gives Fig. 6. The dotted line in this figure illustrates the fact that ROUND also participates in the SO relationship in the present design, because of its inclusion in A#. The participation of ROUND in the SO relationship adds no information that cannot be found from the participation of ACC in this relationship, i.e. an occurrence of the SO relationship concerns only one occurrence in ACC, and this occurrence in ACC concerns only one ROUND. The dotted line represents

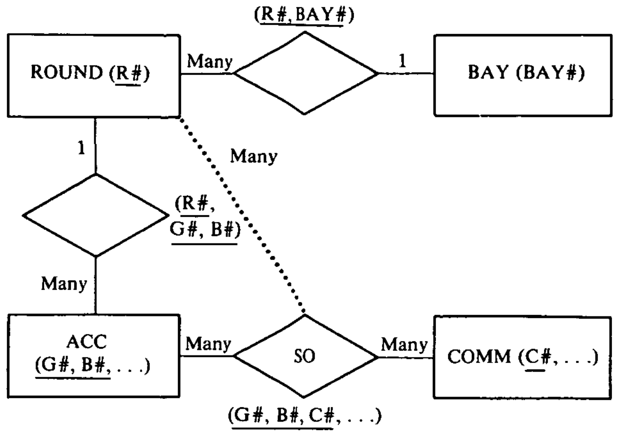


Figure 6. R# extracted.

a tuning decision embodied in the design; storing this relationship may assist processing when the model is physically realized, but it can be left out of our present search for the heart of the matter.

Turning to the commodity sales history, one interpretation of the present record is that it contains the attribute G#, whose value is implicit in the position of the totals. Even if we persist in seeing COMM as fully normalized (by taking G# into the formation of unique names for the attributes), we may now recognize that G# appears in two places—explicitly in ACC and implicitly in COMM. An attribute should appear in only one entity or relationship of the model, with two exceptions: (a) when two separate entities need the attribute in question to form their primary keys, and (b) when the attribute in question is in a relationship and is the primary key of an entity joined by the relationship. This, and the 1:many relationship between groups and branches, strongly suggests that GROUP is an entity which should be accounted for, and that the sales totals for past weeks and months arise in connection with a many:many GROUP–COMM relationship (actually implemented in the case as a 24:many relationship). Figure 7(a) shows a short representation of this, Fig. 7(b) shows it decomposed.

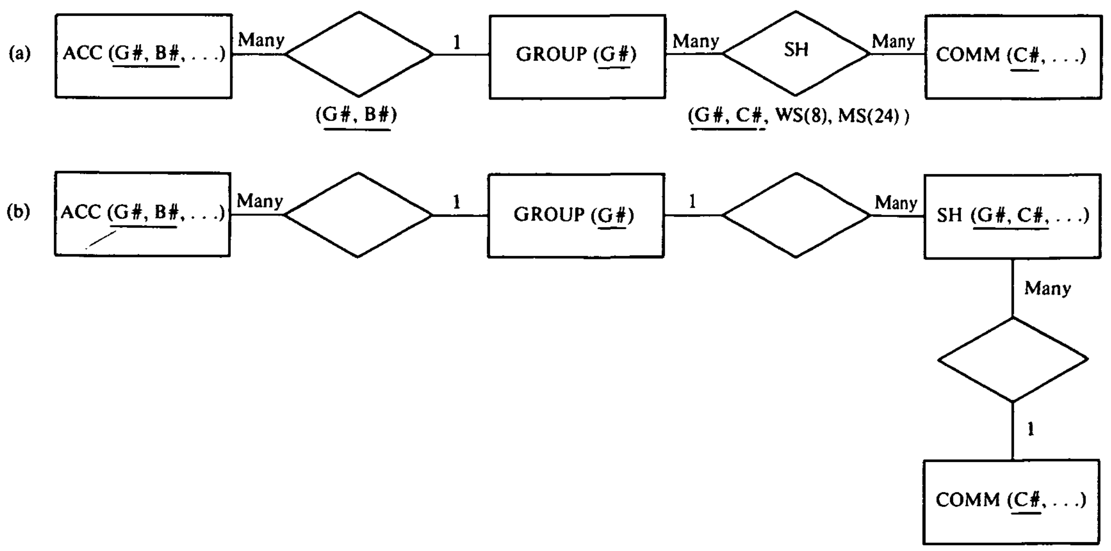


Figure 7. Sales history (SH) drawn out as a GROUP–COMM relationship.

This raises the question of whether or not ACC occurrences with B# = 999 should be stored with other ACC occurrences if our present idea of ACC is that it is a branch involved in ordering and to whom we render an account. Perhaps a more precise view would be that there is a branch, which orders, and an account, which is rendered either for a branch or for a group. There is a many: 1 branch–group relationship, a 1:1 branch–account relationship and a 1:1 group–account relationship; a given account participates in one relationship and not the other. Should we be storing account attributes with GROUP?

There is no great harm in storing B# = 999 entities in ACC and in one sense it keep things simple. However, it might be more in keeping with the facts and the processing if we were to recognize group account attributes as different from branch account attributes. Let us put this on one side for the time being. In other accounting systems I have investigated it has been profitable to split up the TRANS entity into different entities representing different types of transaction. In this case, though, every TRANS enjoys all the attributes and relationships which are to be stored, so they may as well be considered one entity-type.

Finally, let us consider the array attributes which have implicit day indexes in SO and implicit week/month indexes in SH and ACC. A standing order could be conceived as something which connects a SQ, RQ, DQ trio of quantities to an account, a commodity and a day. A representation is given in Fig. 8. The meaning of a relationship like this should always be checked out by substituting an entity for the relationship and considering the relationships between this entity and the original entities. For brevity, this process is skipped here.

If we were to realize the relationship described in Fig. 8 in the most straightforward fashion, we would pay a penalty of storing G#, B#, C#, D# for every occurrence of the trio SQ, RQ, DQ (whereas at present we stand to store G#, B#, C# for every six occurrences of the trio); but we gain by not having to store the trio for inapplicable days (at present we store the trio for every day). Which is preferred from an economy of storage point of view is a question of the facts of the case, which can be analysed as follows. Let x be the average number of days (out of six) for which customers require a type of commodity. Implicit days may be preferred if (total characters stored with implicit days, Fig. 5 with G#, B# substituted for A#) is less than (total characters stored with explicit days, Fig. 8), i.e.

$$10\,900 \times 62 < 10\,900\ x\ (2+3+3+1+3+3+3) \text{ or } x > 3.4.$$

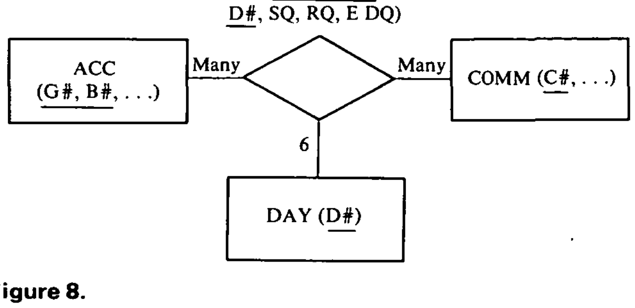


Figure 8.

Let us assume that the implicit representation results in less storage requirement. On the whole, a design which minimizes storage seems to put one in a good position for considering the processes. Only if a processing constraint is struck will it be necessary or desirable to trade some storage for faster processing, and it is very difficult in complex cases to forecast the processing constraints prior to getting down to detailed timings.

Although for this discussion we are putting the days back into SO, it was good to take them out and give them an airing, because of the degree assumption embodied. We must check out with management that a given standing order can concern at most six trading days.

Essentially the same arguments can be made about the implicit week/month index in the sales history, SH. Let us assume that GROUPs and COMMs on average persist long enough to warrant providing for all the weeks and months of all of them. We must also check out that one sales history can at most concern eight weeks and 24 months. Similarly with the previous balances in ACC; one account is to have at most three previous balances stored.

FURTHER ANALYSIS—REMOVE DEGREE CONSTRAINTS, REDESIGN CODES

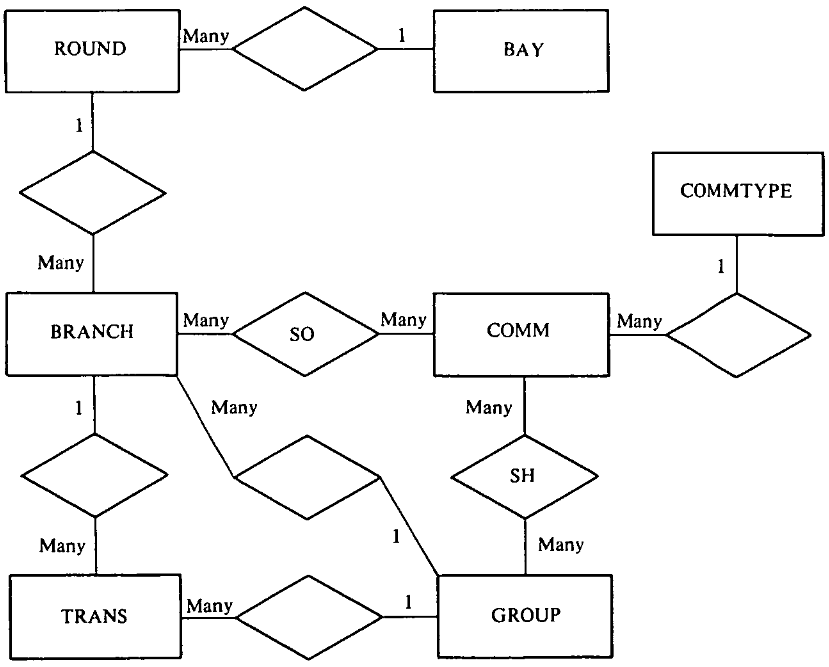
We have already considered that the degree assumption of 25 transactions per account is arbitrary and should ideally be many: 1. Let us take the average number of transactions per account to be 20. A further degree assumption in the present system is that a commodity will have sales history relationships with 24 groups. We know there are only 21 at present, and although new groups may be slow in arising there is no definite limit to the number. Ideally the sales history relationship should be many: many. There are other degree constraints embodied in the value domain of the existing codes (e.g. maximum of 99 rounds) but these seem very much further removed.

Let us now reconsider the code systems of the primary keys of the entities. BAY#, R#, T#, G# are all simple sequence codes and unobjectionable. The hierarchical (G#, B#) to identify ACC is ungainly. We would like to redesign B# so that it uniquely identified occurrences of ACC, allowing us to drop G# from the key. We could do this by assigning a simple sequence code to B#, running independently of G#, but for one fact: we would destroy the flag value information in each existing occurrence of ACC where B# = 999, i.e. we would no longer know which accounts were group accounts. This is a confirmation that occurrences with B# = 999 are unhappy where they are and should be stored in the GROUP relation. A further confirmation that this is a better view is that the B# = 999 accounts do not participate in the ACC–ROUND relationship nor the SO relationship. It is therefore proposed that G# be dropped from the key of ACC, which could be renamed BRANCH, and that GROUP takes up the relevant attributes of ACC (they will be prefixed by G to show that they are the GROUP version of the attributes). The account class attribute, AC, is presumably an attribute of GROUP and not of BRANCH, whereas CNA is presumably an attribute of BRANCH but not of GROUP.

A consequence of this decision is that it is no longer necessary to store G# in the SO relationship. Which GROUP participates in the SO can be discovered through the BRANCH. The shortened key of SO would make it slightly more likely to be worthwhile to store explicit days; branches now need to order commodities for an average of 3.75 days or more.

C# is a block code and therefore conceals an attribute. If we redesign C# as a simple serial code and add a new one-character attribute, commodity type (CT), we increase the record size slightly, but increase opportunity to reduce filesize because there is no need to provide independent growth allowance for bread items. There is not sufficient information to be precise, but let us say that providing for 450 commodities with the redesigned code would give roughly the same growth potential as the existing provision for 99 bread commodities and 400 confectionery commodities. Presumably the designers of the existing system were influenced to build in a growth allowance either because memory management facilities of the day made extension of files difficult, or because the commodity codes which exist at one moment are allocated sparsely over the range, perhaps as a result of rapid change of the commodities on offer.

If sequential codes can be preserved in a close group, this increases their potential for use as keys of relative files. BAY#, R#, G# values all appear to be quite stable. Presumably the B#s in use are also quite stable, but there is more of a question mark over values of C#. The stability of the set of codes in use can be promoted by reuse of defunct allocations, but such reuse needs to be strictly controlled if ambiguities are to be avoided. A suitable rule for B# here could be that a B# may not be reused until after the existing occurrence with that B# is deleted from BRANCH; and an occurrence of BRANCH may not be deleted until there are no transactions for that branch and the account balance is zero. This will mean that the B# of a defunct customer could be reused, at the earliest, two months after the last transaction with him. A C# should not be reused until after the existing occurrence of that C# is deleted from COMM, and this deletion should not be permitted until all SH occurrences for that C# have zero sales. This means that a defunct C# is not available for reuse until two years after the last delivery involving that commodity. Figure 9 shows the state of the data model after incorporating the changes discussed in this section.



BAY (<u>BAY#</u>)
BAY-ROUND (<u>BAY#,R#</u>)
ROUND (<u>R#</u>)
ROUND-BRANCH (<u>R#, B#</u>)
BRANCH (<u>B#</u>, B%, C%, INA, CNA, CB, PB(3))
BRANCH-TRANS (<u>B#, T#</u>)
TRANS (<u>T#</u>, TC, TD, T£, TF)
BRANCH-GROUP (<u>B#, G#</u>)
GROUP (<u>G#</u>, GAC, GB%, GC%, GINA, GCB, GPB(3))
SO (<u>B#, C#</u>, SQ(6), RQ(6), DQ(6))
COMM (<u>C#</u>, PQ, P£, PP, PD)
SH (<u>C#, G#</u>, WS(8), MS(24))
COMM-COMMTYPE (<u>C#, CT</u>)
COMMTYPE (<u>CT</u>)
TRANS-GROUP (<u>T#, G#</u>)

Figure 9. The primitive model, stripped of tuning relationships.

CHECKING OUT THE STORED COMMODITY TOTALS

It will be recalled that we put aside the question of whether or not the commodity totals stored during production of the delivery notes should be included in the model. Perhaps we now have sufficient overall grasp of the facts to tackle this question of detail.

At the end of the delivery note run, the stored totals represent the accumulated quantities for each commodity, within each round, within first or second delivery. These stored totals are used to produce the two reports that summarize the deliveries: the van loading report and the bakery loading report. For this purpose, the stored totals are simply a derivation of data stored in accordance with the model, and therefore can be considered a workfile. However, the case description states that the delivery note run relates to deliveries of the next day's bread, but the next-but-one day's confectionery. Obviously the delivery note lines for two different days cannot be intermingled. It would be possible to mingle bread and confectionery lines relating to the same day, if this were desired, if there were two passes of the standing orders file per day, one being a read-only pass to forecast the day after next's confectionery for van and bakery loadings, and one updating pass to produce the next day's confectionery and bread delivery notes, followed by the van and bakery loadings for bread. The confectionery notes produced in this way would be the ones that related to the standing orders included in the previous day's forecast; to preserve data integrity, the system would have to reject any attempt to amend the required quantity after the forecast had been made. But this cannot be the procedure because it would not, as the case insists, then be necessary to store the round/delivery totals from one day to the next.

We must conclude that only one pass of the standing orders file is made, confectionery delivery notes being produced separately from bread delivery notes. Let us assume this to be the correct interpretation, and that confectionery notes are required a day earlier than bread. Presumably, what I have just described as one pass is in fact two sweeps of the standing orders file, all the confectionery notes being produced on one sweep, all the bread notes on the other. This means that when the van driver gets his delivery notes in the morning, he gets the bread notes prepared the day before and the confectionery notes prepared the day before that. Similarly the bakery manager gets bread loadings prepared on the day on which the evening shift starts, but confectionery loadings prepared a day and a night before the day shift starts. The delivery note run is done in two parts: one to produce all the delivery notes, van loading and bakery loading for bread items to be delivered tomorrow, and one to produce all the delivery notes, van loading and bakery loading for the confectionery items to be delivered the day after tomorrow. The notes produced from the second part on a given day need to be held and married with those produced from the first part on the next day.

In order to know where to break the deliveries, when producing the bread delivery notes on a given day, the accumulated pack points for the confectionery deliveries prepared the previous day must be available. Although these pack points were a straight derivation at the time they were stored, are they still a derivation by the time they are re-input the next day? The pack points could not be rederived from the required quantities if the standing quantities were used to update the required quantities in the manner previously explained. This is because a temporarily required quantity used in the original pack points calculation will have been restored to the standing quantity.

If this account is correct, it follows that the accumulated pack points per round from the last delivery note run is an attribute which should be accounted for in the model. The pack points are an attribute of the round—ROUND (R#, BAY#, B/F CONFECTIONERY POINTS). (Even if this attribute is technically a derivation, because it may be the aggregate of the recorded delivered quantities, there is no harm in including it in the model if it helps understanding.) This point is perhaps a little academic, but it has brought out a feature which is unclear in the present design. Presumably the existing system works something like this: the bread delivery notes are produced before the confectionery delivery notes, the bread phase of the program re-accumulates the pack points from the stored totals, the bread delivery notes etc. are produced, the stored totals are set to zero, then the confectionery notes are produced, updating the stored totals.

It is not surprising that some manual adjustment is needed when making up the van loadings. There are a number of interesting speculations about how well the delivery system works, but once again in the interests of a brief discussion the functional targets of the system will be taken as given.

This point also reveals a rather fine system anomaly. Suppose the ROUND of a BRANCH changes. The B/F CONFECTIONERY POINTS (or stored totals) for the old round will include points (quantities) which should be in the points (stored totals) of the new round. Similarly, the confectionery delivery notes and van loading will be for the old round, the bread delivery notes and van loading for the new round. This may make the delivery break more inappropriate than usual, and may entail a bit of juggling with the delivery notes; perhaps little more than the usual manual adjustment. A more perplexing possibility is that the new round belongs to a different bay. If the system functions normally, the confectionery products for the customer will be delivered to one loading bay and his bread products to another. The van driver, probably having in his head the round change and the bay associated with the previous round, may yet be able to untangle this. This anomaly could be avoided by storing a next-round relationship between ROUND and BRANCH. The confectionery notes would be produced using next-round, if present, then the next run would use the next-round to update the current round for the bread delivery notes. Let us assume this is not necessary.

REALIZATION OF THE MODEL

A set of files derived from Fig. 9 in the most straightforward manner is as follows:

ROUND(R#, BAY#, B/F CONFECTIONERY POINTS)
BRANCH(B#, R#, G#, B%, C%, INA, CNA, CB, PB(3))
TRANS(T#, B#, G#, TC, TD, T£, TF)

GROUP(G#, GAC, GB%, GC%, GINA, GCB, GPB(3))
SO(B#, C#, SQ(6), RQ(6), DQ(6))
COMM(C#, CT, PQ, P£, PP, PD)
SH(C#, G#, WS(8), MS(24))

In the interest of brevity, details of the redesign of the system procedures using the new files are omitted. Interested readers can obtain this detail from the author should they desire.

The conclusion of the redesign is that filestore requirements are reduced to 2 668 304 characters approximately. The original daily schedules program, which of all the programs probably used the most computer resource and had a deadline with little slack, had an estimated 15 000 disk accesses with an average seek of 12 cylinders. Redesigning this program using the new files, but for the same moving-head disk assumed for the original program, gives a revised estimate of 19 000 similar disk accesses with an average seek of only 1½ cylinders. Probably the disk I/O resource usage of both versions is similar even without making any tuning adjustments to the redesign. Probably both old and new versions of the program would be 'printer-bound' on the assumed equipment.

Should tuning of the daily schedules program be thought necessary, maintaining an alternate index to the BRANCH file on the attribute R# eliminates 6000 retrievals which arise from the need to search for branches on the current round. This tuning shows a clear advantage without complicating the design. Alternatively, the BRANCH file can be efficiency-split into BRANCH-DEL(B#, R#, CNA) and BRANCH-ACC(B#, G#, B%, C%, INA, CB, PB(3)). Only BRANCH-DEL is needed for delivery notes and this allows a larger blocking factor, roughly halving the 6000 searching retrievals.

The stored totals workfile contained limitations because of its assumptions about the maximum number of rounds, deliveries and commodities. These assumptions are eliminated in the redesign, without significantly increasing computer resource usage, by writing a scratch sequential workfile with records (CT, R#, DELIVERY#, C#, RQ) which is then used separately to produce the van and bakery loadings.

In the redesign, all the master files are left in primary key sequence, which bodes well for simplicity of file maintenance, and efficiency and flexibility in meeting new requirements.

CONCLUSIONS

Estimating resources used by computer runs is error-prone, and doubly so when not all the pertinent facts are known about the hardware, software and process characteristics. However, it seems likely that the tuning assumptions embodied in the original design were not justified. The data analysis has created a track through the jungle of detail, made tuning decisions explicit and given a basis for comparing alternatives. The data modelling has reduced redundancy a little and reduced wasted space. As a result the starting requirements for online filestore have been substantially reduced. Disk I/O resource usage of seeking, latency and transfer can be reduced if this is a target. Processor overhead is probably much the same.

The original design had limitations if more than three new groups arose, if rounds ever exceeded 45, if bread commodities (existing at any time in a two-year period) exceeded 99, if confectionery commodities exceeded 400, if an account had more than 25 transactions, if a round had more than two deliveries, if some groups opened more than one new branch, if a new commodity type was required. These limitations, except the last, have now been postponed to the limits of filestore capacity or the limits of the value domains allowed by the primary keys. To allow additional commodity types, the many:many discount relationship between GROUP or BRANCH and COMMTYPE would have to be explored, as well as possible many:1 COMMTYPE–ROUND relationships concerning pack points.

The original design gave rise to processing complexities, or data anomalies, if a branch changed its round (as a result of management decision) or a commodity changed its type (as a result of reclassification). These complications have been removed.

We do not know where the ROUND–BAY relationship was stored in the original design, but this has now been drawn out as a file, thereby parameterizing the van and bakery loading reports and facilitating loading bay changes. The data analysis encouraged detection of a deep-seated anomaly which would occur when a change of the round of a branch also gave rise to a change of loading bay. The need to check out degree and other assumptions gives direction to the fact-finding and improves the reliability of the design.

Data analysis may affect program complexity. It encourages the use of simple files containing fixed-length records of only one type. It leads to more files per program, and perhaps to the use of search algorithms that are not so familiar to programmers. Nevertheless, there does not seem to be an inherent difference in complexity when compared with traditional designs. Quite plain algorithms tend to result if there is suitable software support for alternate indexes.

REFERENCES

1. P. P. Chen, The entity-relationship model—towards a unified view of data. *ACM Transactions on Database Systems* **1** (No. 1), 9–36 (March 1976).
2. A. Parkin, *Systems Analysis.* Edward Arnold, London (1980).
3. H. D. Clifton, *Data Processing Systems Design.* Business Books, London (1971).

Received June 1980