Working Toward Step One,
a method for deriving your 4D file structure

Lincoln Stoller, Ph.D.
Braided Matrix, Inc.

Introduction

Database programs perform many functions, and these functions tend to evolve over time. Consequently, it's especially important for those who design databases to have a clear vision of their project before they begin to program. In spite of this, many microcomputer programmers are unfamiliar with the software design techniques. In addition, many of the techniques that are available reflect their academic origins by emphasizing a theoretical approach that hinders their usefulness in the prototyping process.

Here I elaborate a method offered by Steve Eckols in his book How to Design and Develop Business Systems. I do this by applying the method to design a basic accounting system which, if not widely familiar, is important in many applications. My task is to draw the 4D file structure required to meet the needs of a system that handles general accounting entries as well as Payables and Receivables. This job ends when I've laid out the files and relations that appear in the 4D structure window, just at the point where one sits down to program.

Eckols' method involves looking at the design problem from predefined angles and corralling the problem, animal that it may be, with insights and specifications drawn from each. This allows you to work from the system as it initially appears and to build the database system incrementally.

For purposes of explanation I'll present the method according to the following sequence of steps:

A) The creation of the *data flow diagram*, the major overview document that shows what processes need to be accomplished and what information needs to be stored.

B) The creation of *contents lists*. These documents suppress processes and data flows and provide an opportunity to scrutinize the system's information structure. They add detail to the information storage requirements sketched in the data flow diagram.

C) The system structure tools of section 4 suppress the data structures considered in the contents lists and focus on the processes outlined in the data flow diagram. These tools consist of the *system structure table*, summarizing the processes that the system performs on each data store, and the s*ystem structure chart* portraying these processes as interconnected modules and mapping the user's path through them.

D) Finally there is the data access analysis described in section 5 that offers a method for deducing the file structure from the contents list and the system structure chart. This part of the analysis requires you to list the data attributes needed to identify the tasks enumerated in the system structure chart and place them next to the data stores given in the contents lists. You assign keys to data stores, identify files and clarify one-to-many and many-to-many relationships. As a schematic the *data access diagram* shares some of the properties of an entity-relationship diagram (see Elmasri 1989, chapter 3).

I won't go through the critique and revise cycle that is an inevitable part of incorporating new insights. I'm also leaving aside the formal rules of database normalization although we will incorporate normalization at a common sense level along the way. (See W. Kent's article [W. Kent 1983] or one of the many more recent texts for an introduction to the normalization process.)

If, while I'm developing this prototype, you can't see your problem standing at attention with the clarity of the following example, do not be discouraged. In practice this process requires several cycles, each cycle improving upon the diagrams of the cycle before it. Once you understand the strategy your tactical skills will improve and you'll reach a solution more quickly.

The database problem and the Data Flow Diagram

The problem is to create the 4D file structure of a basic accounting system for Braided Matrix, Inc. that provides:

- a list of accounts,
- double entry transactions that record the balanced flow of assets into and out of accounts,
- receivable and payable account management that allows me to match funds received with items remaining in receivable accounts and funds paid with items in payable accounts,
- a variety of summary accounting reports.

The Data Flow Diagram

The first step is to picture the problem in terms of the processes that take place and the data stored. The data flow diagram (DFD) shown in figure 1 is comprised of external elements called terminators lying at the periphery itemizing events that begin or end processes. The terminators connect to processes which can either lead to or join with other processes. Alternatively, they can draw from or write to stores of data.

In figure 1 the external elements are set off with single lines, the processes with circles, and the data stores with 3-sided boxes. The arrows depict the flow of information from a process to a data store or from data store to process.

Moving clockwise around the diagram are the following functions:

Enter double entry transaction: Accounting transactions record the movement of assets, liabilities and equities, and are composed of 2 or more components giving the amounts moved to and from various accounts. Entering a double entry transaction requires locating the accounts involved from the Accounts store and saving a summary of the transaction and its components to the Transaction data store.

Modify the list of accounts: Accounts represent clients, suppliers and departments within Braided Matrix's organization.

Match receipts with amounts due: Processing receivables involves matching a credit granted to a client, as entered through one transaction, with funds received from the client and entered through a later transaction. Although I don't go into detail, it's important to have an idea of what needs to be done to manage receivables. In this regard we'll examine two transactions that complete a receivable entry.

The first transaction consists of two components, a debit of $2 to a client's receivable account, say Apple's, and a credit of $2 to an income account that tracks monies earned by Braided Matrix. The second transaction is entered when a $3 payment is received from Apple. One of its components is a $3 debit to the Walt Disney National Bank where the money is deposited; the other is a $3 credit to Apple's receivable account.

Once the two transactions have been entered managing this receivable item involves selecting Apple's account and locating the two components of the transactions mentioned. (Remember: two transactions generate four components, but only two components involve Apple's account.) Then I retrieve the amount owed, $2, and "cover" it with the amount paid, $3. The original amounts debited and credited to Apple don't change but the components do record that of $2 owed nothing remains due, and of the $3 paid $1 is left to cover other receivables.

Match payments with obligations: Processing payables is essentially the reverse of receivables.

Print reports: Reporting should be isolated from other operations as much as possible in order to simplify the DFD. In the design stage, reports provide convenient lists of the data that you need to store and track. To a large extent, once this data is in the database reporting is just a process of extracting, formatting and printing it. When this is true, as it is for the reporting tasks diagrammed with shaded arrows in Fig. 1, reporting does not impose additional requirements on the data and can be ignored until after the data structure is set up. For this reason we can consider reports and reporting peripheral to data flows; reports can easily be added to the final project.

The DFD is a working document that provides a comprehensive view of the project; we need not distinguish every procedure or resolve every element to the same level of detail.

The arrows used in the DFD represent data flow in a broad sense. They can be thought of as replies--supplied by the data stores--to questions asked by the processes, or as data entered by the user as part of a process. The arrows don't signify hierarchical relationships and the processes they connect don't need to follow in rigid sequence. Nor do they indicate unique means of access to the various data stores. These aspects of the database are represented in the system structure table, system structure chart and the data access diagrams described later.

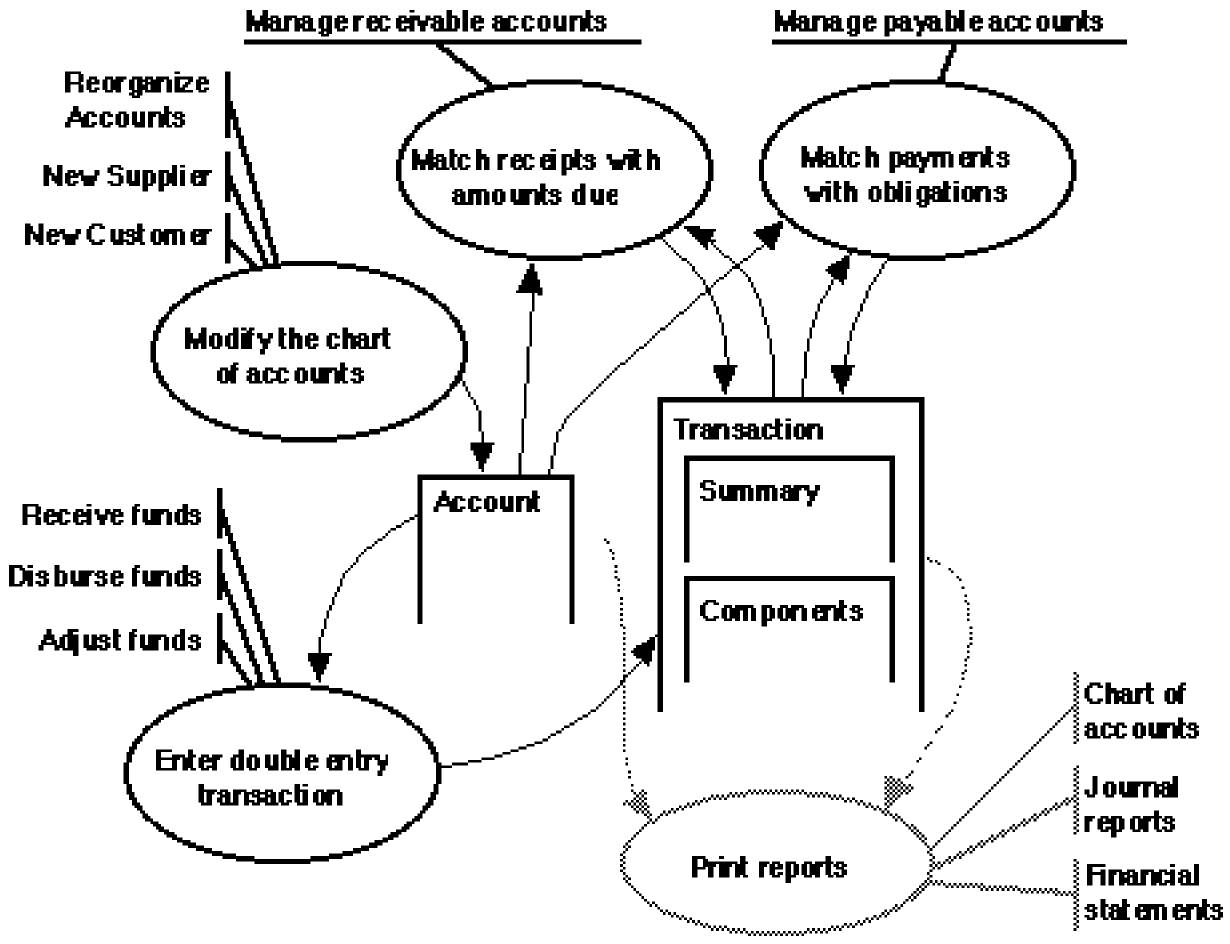


Figure 1: the Data Flow Diagram

The data store labeled TRANSACTIONS contains the SUMMARY and COMPONENT data stores. This nesting exemplifies the way elements of the DFD may contain further structure. The encapsulation of the SUMMARY and COMPONENTS file is a short-hand illustrating that for some purposes transactions can be considered as a single entity even though it's really composed into two separate stores.

The composite structure of the TRANSACTIONS store follows from the following properties of SUMMARY and COMPONENT data:

• the number of components is unlimited in theory so there is no effective way to reserve space for component information as part of the summary.

• in order to report on componets on an account-by-account basis, instead of on a transaction-by-transaction basis I need to gather one or more components from many transactions. This will require including some components from a given transaction while excluding others.

Because of this independence between summary and component data they are assigned to separate data stores. So even though components are always bound to summaries when their data is entered the reporting and analysis of the data requires that components be considered as entries in their own right. This exemplifies the importance of considering the full use of the data before deciding on its structure.

The Contents Lists

The contents lists display the unstructured “atoms” of information that the database will manage. The lists are created for each data store independently without any consideration of relational structure or data duplication. The items in the lists are taken from the forms and reports that the database is required to produce and the usually similar lists of input information. Items that are internal to the operation of the program, such as key fields that enforce unambiguous relations between different files, will be considered when I complete the key field chart.

The contents lists consist of a set of tables, one for each data store, listing everything that needs to be found or referenced through that store. The elements are specified according to "group," "repetitions" and "type."

The group column distinguishes composite fields. An item may appear as an element in the contents list even if multiple instances of that item occur. Such elements are called subordinate structures and are marked with a "Y" in the group column. In terms of the rules of normalization, splitting off subordinate structures puts the data structure into 1st normal form.

In this example COMPONENT’s are a subordinate structure; there are several components to a transaction (at least two) and each have the same structure.

The repetitions column shows the number of elements that can be assigned or related to an instance of the data store. The number of repetitions can be a single number, such as 1 when the field is mandatory, or can range between the least and the greatest number of instances.

In the type column we put the 4th Dimension type of each nonsubordinate component.

Data Store:	Transaction: Summary		
Element	**group**	**repetitions**	**type**
title		1	alpha 40
document number		1	alpha 10
COMPONENT	Y	2-n	
entry date		1	date

Data Store:	Transaction: Component		
Element	**group**	**repetitions**	**type**
account name		1	alpha 40
account number		1	alpha 10
debit flag		1	boolean
amount entered		1	real
amount outstanding		0-1	real
description		0-1	alpha 80

Data Store:	Account		
Element	**group**	**repetitions**	**type**
account name		1	alpha 40
account number		1	alpha 10
account type		1	alpha 20
current credit balance		1	real
description		0-1	text

Figure 2: Contents Lists

The data items don’t need much elaboration. Summaries are given a title, number and date and contain two or more components. Each component records an amount that’s either a debit or credit as well as an account name and number. If the component records an amount due to be paid or received then it will also store the total amount still owed and still to be received. Components that are not due for payment or receipt, such as expenses, deposits, withdrawals and the like, don’t use of the “amount outstanding” field.

Accounts are given a name, number and description and store the cumulateve total of all monies credited minus all monies debited to them. In addition, the rules of double entry accounting require that every account be either an asset, equity or liability accout. The field “account type” is assigned one of these three labels.

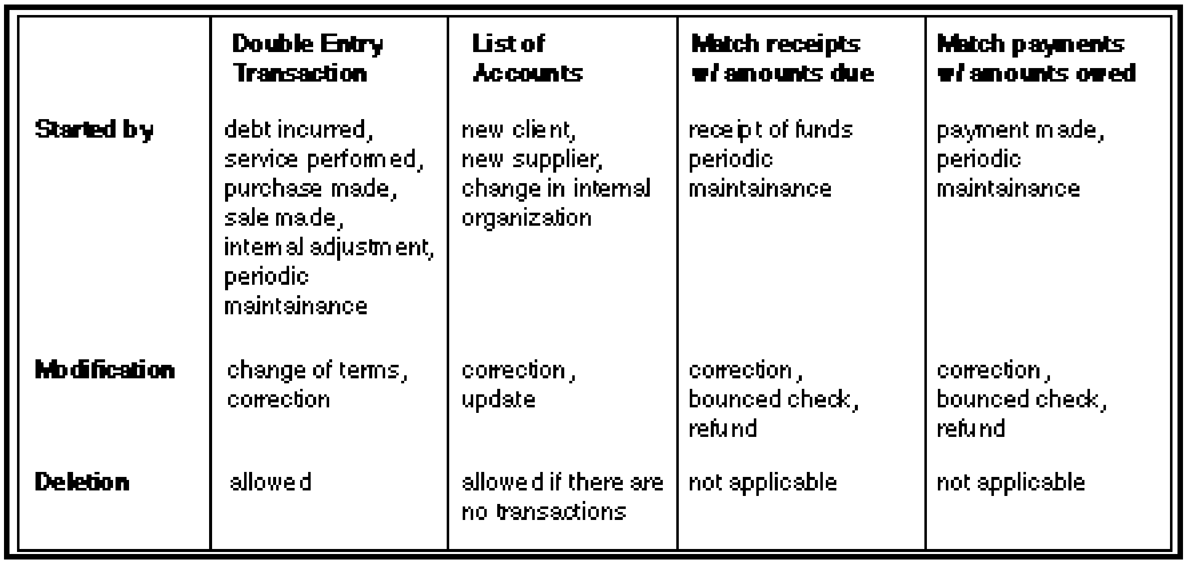
For the most part the data stores will correspond to the files of the database; the contents lists telling what information that needs to be retrieved in each case. Because the contents lists don't tell you where the information should be stored the same item (account name for example) appears in different data stores.

Inevitably, some of the information in one data store will be retrieved from other data files; it's even possible for a whole data store to consist entirely of information drawn from other stores. This is, in fact, just what happened with reports and is the real reason we could dispense with them early on. If I had not eliminated the reports then at this stage I would have a store for each report. It would then become apparent that every item of the report files is drawn from another, nonreport data store. By eliminating reports we are left with a structure in which each data store turns into a separate data file.

Deciding how to split up the information shown in the contents lists and how to establish the relations between data files follows from the data access diagram. In order to derive this diagram I need the system structure chart which I consider next.

The System Structure Table

The system structure summarizes the actions that the system performs from the user's point of view. The system structure table breaks down the major functions of the data flow diagram acording to their starting actions, modifications and deletions. This forces us to consider and settle basic processing issues before proceeding. It also highlights similarities enabling us to combine different actions into general structures. In some cases we can even merge different functions as occurs below.



	Double Entry Transaction	**List of Accounts**	**Match receipts w/ amounts due**	**Match payments w/ amounts owed**
Started by	debt incurred, service performed, purchase made, sale made, internal adjustment, periodic maintainance	new client, new supplier, change in internal organization	receipt of funds periodic maintainance	payment made, periodic maintainance
Modification	change of terms, correction	correction, update	correction, bounced check, refund	correction, bounced check, refund
Deletion	allowed	allowed if there are no transactions	not applicable	not applicable

Figure 3: System Structure Table

From the similarity displayed in the handling of payables and receivables, together their identical domains of operation as seen in the arrows of the data flow diagram, we can consider payables and receivables as two cases of one operation that we'll call "allocation of funds." We'll combine these two elements in the system structure chart and let the graphical user interface distinguish the two types of allocation.

The System Structure Chart

The system structure chart (SSC) suppresses all reference to the data stores and other internal structures. In doing so it provides a tool to visualize how tasks are grouped and to decide whether their groupings are consistent with their functions. The SSC arranges the system's functions into modules of related actions in a tree structure. These modules represents the sequence in which these functions are performed. I have used separate modules to represent unrelated functions or functions to which access is restricted. The resulting diagram shows the user's options as they'll appear in successive entry screens.

While the construction of the user interface is outside the current discussion you should note that the SSC provides a context for analyzing how the user interface interacts with the structure of your database. When completed, the SSC is a blueprint for the user interface.

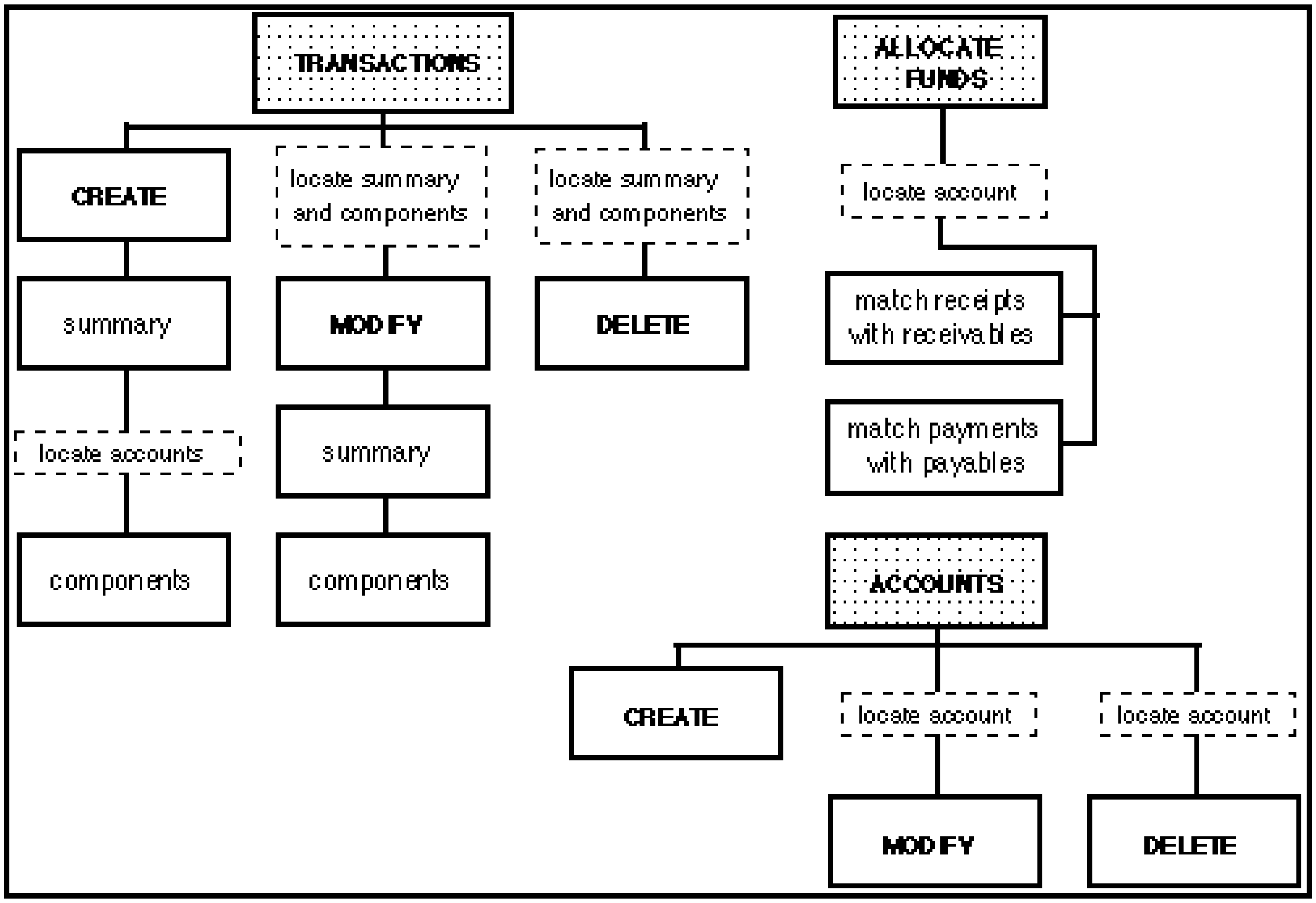


Figure 4: System Structure Chart

<u>The Data Access Diagram</u>

The process of drawing the data access diagram turns data stores into data files relationally linked by key fields.

To construct the DAD I first create a key field chart by setting the contents list and the system structure chart side by side. I draw and label a box for each data store in the contents list. Then I go through the system structure chart, branch by branch. I make a list of all the fields it involves in each process depicted by a box. These are the fields that will be used by that process to access information from the data stores. I connect these fields to their respective data stores using either a plain line or a line that ends with a small box. If one field value can be related to more than one record of the data store I use the line that ends in a box. If the field uniquely identifies a particular instance of the store I omit the box at the end of the line.

The result for the Braided Matrix accounting system is shown in figure 5. Since every field that keys to the Component store also keys to the Summary store I continue to enclose these in a Transactions box.

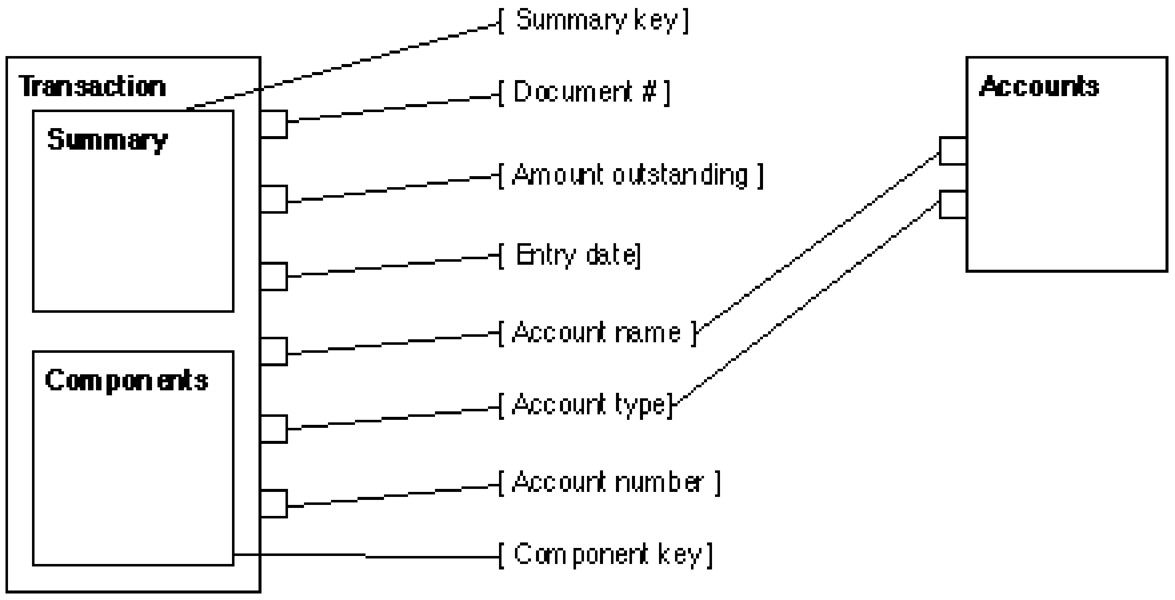


Figure 5: Key Field Chart

The next step is to associate fields with data stores. This is done according to the following guidelines:

- If a key field is connected only to one store it should be moved into the box representing that store.

- If a key is connected to several stores but is uniquely connected with one, it is moved into the store with which it is uniquely related. Preserve the lines drawn between the field and its other related data stores.

- The fields associated with composite stores, such as Transactions, are placed in the component store with which they are most closely associated.

Once a store contains at least one field it is elevated to the status of a data file; that is, once a data store becomes part of the file structure it is no longer just a functional concept.

These guidelines generate a file structure with the required data items and access paths. However they don't guarantee a consistent (normalized) structure. Additional care needs to be taken to be sure that the structure generated is storing data efficiently. In this regard there are some cases worth special mention:

• A key that's uniquely associated with more than one file indicates two files that should be incorporated.

• Keys accessing compound structures should be placed in files with which they're most uniquely associated.

• A key that's not uniquely associated with any file is a data structure in itself and may require a separate file. This is the case with the account type and journal keys. Since these keys are not modified and carry no additional information they are safely stored in 4th Dimension lists.

•If a file has no unique keys you may be unable to distinguish one record from another and you should add a unique key to the data access diagram and contents list. This is the case for the Summary and Components files. Keys added in this manner are strictly internal to the function of the database. The database application automatically assigns values to these internal fields according to rules that enforce the integrity of the data. The values assigned to the internal fields and may or may not be seen by the user

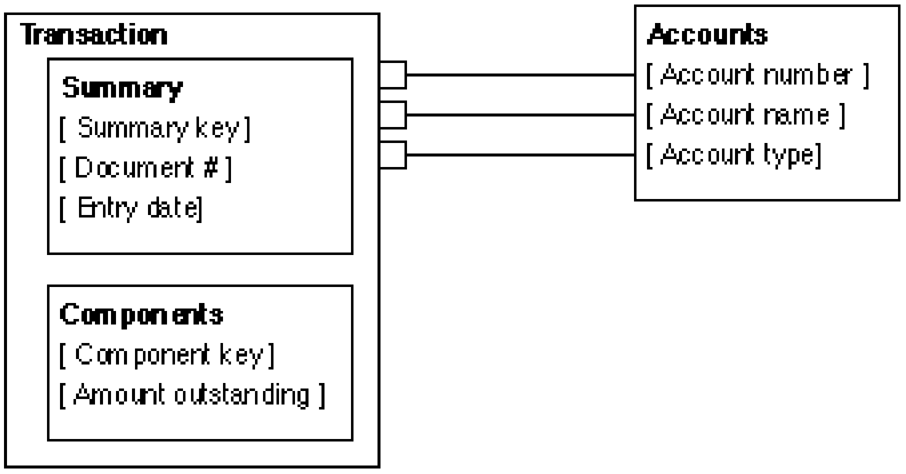


Figure 6: Data Access Diagram

<u>4D's File Structure</u>

To reach the 4D file structure of figure 7 you need to recall that two files only need to be linked by a single field in order for one file to be accessible through the fields of the other. This means that we can replace the three-fold links between Accounts and Components with a single relation.

A 4D file relation is established by drawing a file relation line between two files. A field must exist in both files that contains the same information. The file that the field identifies uniquely is referred to as the one-file. The file that is not uniquely defined by the field value is called the many-file. The file relation line is drawn with its arrow pointing from the many-file to the one-file.

The subordinate relation of the Components file to the Summary file is realized by linking a field in the Component file to the summary key field. To do this I add a field to the Components file that references, that is takes the same value, as the key field of the Summary file. A similar subordinate relationship exist between Components and Accounts and requires a second field to be added to the Components file to act as a pointer to Accounts.

The fields involved in the 4D file relation must be indexed. The fields distinguished as key fields of the DAD are also indexed since they provide values for frequent searches and sorts. The 4D structure shown in figure 7 also includes the remaining fields from the contents lists.

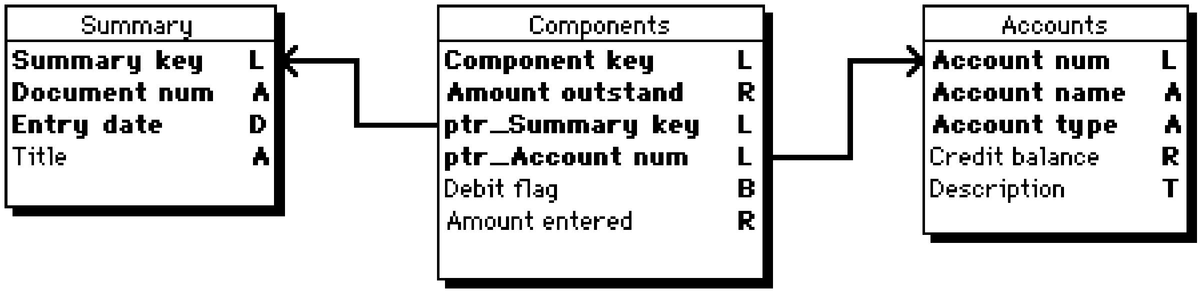


Figure 7: 4D File Structure

Conclusion

With the creation of the 4D structure our work is complete. The first half of what I did specified the system. The data flow diagram, contents lists and system structure each added information from the complementary perspectives of system overview, system structure, and system function. The second half was deductive, reducing the system specifications to reach a supporting file structure.

We've gone through this process sequentially, while in theory these steps should be done in parallel. But since our minds don't really work in parallel we do the next best thing by repeatedly cycling through the steps. At first we produce faulty sketches that reflect our first impressions and with each pass we add detail and consistency.

In weighing the value of the method against the effort required to complete it you should consider the following benefits:

- the data flow, system structure and contents lists reduce a complicated system down to three comprehensive perspectives,

- the data flow analysis leads you directly from these specifications to a file structure,

- the diagrams and tables that you produce form a blueprint for programming the application,

- the specifications form an archival record for programmers who may work on the project in the future and provide a basis for creating a user manual.

The most impressive aspect of Eckol's method is how it derives a file structure from a carefully paced description of the system's function. Try it just to see how neatly this result pops out.

Bibliography

C. J. Date, **An Introduction to Database Systems, Volume I**, Addison-Wesley, 1990. See part III on the relational model and part V on database design.

Steve Eckols, **How to Design and Develop Business Systems**, Mike Murach and Associates, 1983

Ramez Elmasri and Shamkant B. Navathe, **Fundamentals of Database Systems**, Benjamin/Cummings 1989.

W. Kent, "A Simple Guide to Five Normal Forms in Relational Database Theory." Communications of the Association for Computing Machinery **26**, No. 2 (February 1983)