The structure of accounting systems: how to store accounts

Lincoln Stoller, Ph.D.

balances

In my most recent article I considered the simplest accounting system consisting of a single file of general ledger accounts and another file of transactions. In this article I consider more useful account file structures and review what relational database theory has to say about them. The topic should be of interest to anyone working with accounting data.

As I discussed before, business accounting boils down to the recording of transactions that move assets from one account to another. To whatever extent a transaction increases some account balances, it must decrease others. At all times the totals of all the positive account balances must equal the total of the negative account balances. This is expressed in the first fundamental equation of accounting:

Total Credits = Total Debits

This equation originates from the underlying idea that accounting is the process of tracking valuable “stuff.” It is not so much a rule as a statement that this “stuff” always comes from one account and goes to another.

You can view accounting as having a passive component, which consists of accounts, and an active component consisting of transactions. The transactions act like messages passed between accounts. The messages say how much to change the account balances. Once the account balances change to reflect the transaction, the transaction is finished and has no further effect.

This is a useful view from a relational design perspective because it offers a way to think of the account structure as separate from the transactions. The account structure represents the state of the business after all the messaging is finished at the end of the day or the end of the year.

We can limit our concern to the kind of relational structures that best suit the task of reviewing and extracting information from accounts. This is the kind of task required of bookkeepers, accountants, auditors, managers, and executives. The flip side of the picture focuses on transactions — how to manage them and how to optimize their communication with accounts. I will consider this issue in a future article.

accounts

All accounting systems rely on what's known as the general ledger, which consists of a set of accounts that reflect the structure of the business. Common general ledger accounts used by most businesses include Retained Earnings, Long Term Debt, Cash, Taxes Payable, Advertising Expense and others. Less common general ledger accounts might include Sales Commissions, Foreign Taxes Paid, and Brokerage Fees. In fact, any category that materially affects a business will be assigned its own general ledger account.

The purpose of the general ledger is to present a complete and accurate picture of the business. The picture is complete because it consolidates various related operations under each general ledger account label. The picture is accurate because general ledger accounts should only be updated after any errors that might have existed are corrected, and any temporary or short term uncertainties resolved.

Complementing the general ledger is a separate set of what might be called "working accounts." These are used to collect information before it's passed on to the general ledger. These accounts are variously called ledger, journal, or subaccounts. Whereas the general ledger accounts reflect the overall heath and progress of a business, the subaccounts reflect the day-to-day operations.

There are endless possibilities for business-specific subaccounts. A small computer consulting business would have Computer Hardware Expense, Office Rent, and Professional Dues subaccounts. An entertainer would have a subaccount for costumes, a landscaper a subaccount for shrubs, a truck driver a subaccount for gasoline. There would be separate payable subaccounts for suppliers and separate receivable subaccounts for clients.

Subaccounts feed into the general ledger in the sense that any change in a subaccount's balance must have the same effect on the balance stored in the general ledger. Subaccounts are in a many-to-one relationship with general ledger accounts with the condition that there must be exactly one general ledger account related to every subaccount,

It is not necessary that every general ledger account be related to a subaccount. You can design general ledger systems where only certain general ledger accounts are linked to subaccounts. Such systems will have the following properties:

- They must support transactions that directly effect general ledger accounts.
- The set of subaccounts in such systems will not present a complete picture of the business, since they can be bypassed by some transactions.

In the systems I design every general ledger account is linked to at least one subaccount. This simplifies transaction entry and provides an additional means of checking data integrity. The following entity-relation diagram shows least one subaccount related to every general ledger account. The small box at the end of the line leading to the subaccounts indicates many subaccounts can be related to a single general ledger account. The numbers in parenthesis give the minimum and maximum elements that can be related. In this case exactly one general ledger account is related to at least one, and as many as n subaccounts (where n is any number greater than one).

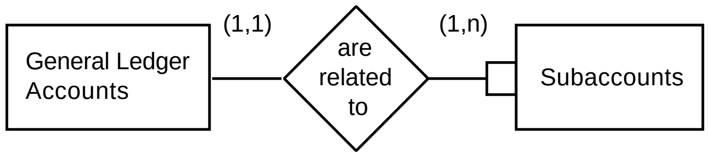

Figure I. An entity-relation diagram describing the relationship of general ledger accounts to subaccounts.

It may seem like subaccounts and general ledger accounts are redundant but that's not so. Subaccounts present a detailed picture and are used in the course of daily business operations. General ledger accounts show long term trends and present an overall picture of the business and its environment.

The two structures also differ in how they're managed. Subaccounts are kept up to date on a daily or real-time basis — general ledger accounts are brought up to date less frequently, perhaps on a monthly basis. The process of carrying over the subaccount balances to the general ledger is called posting. Once all current subaccount balances are posted the general ledger and the subaccounts and the general ledger are synchronized.

General ledger and subaccounts are part of every bookkeeping system. They present the same information from two different perspectives. You could say that the general ledger accounts show us the forest, while the subaccounts show us the trees.

categories

In addition to having a name and a balance, each account is assigned a category. The fundamental account categories are:

- assets,
- liabilities, and
- owner's equity (or equity, for short).

In most accounting systems you will see additional categories such as income, expense, sales, and purchases. These are just particular sub groupings of the basic three used to make bookkeeping easier. However, additional categories will just complicate things for us, so we'll limit ourselves to the big three: asset, equity, and liability.

Roughly speaking, assets are resources owned by the business. They can be tangible, like land or inventory, or intangible, like a patent on an invention or a payment that remains due from a client.

Liabilities are debts. That is, they are obligations the business has to its creditors. This includes tangible things, like cash loans, and legal obligations like taxes payable. You can think of creditors as co-owners of the business because they have the legal right to force the sale of the businesses assets in order to secure the payments owed them.

Equity is the portion of the business assets owned outright by the business or its owners. It's the portion of the valuable resources of the business that would be left over if all creditors demanded immediate payment of all debts.

With these three categories we can express the second fundamental equation of accounting:

Assets = Equity + Liabilities

This says that the value of the resources (assets) is the combination of what's owned outright by the business (equity) plus what's owed to the creditors (liabilities). Like the first equation, which states that debits equal credits, this too should be viewed as a statement of the obvious. It says that all the valuable "stuff" that makes up any business is owned by somebody.

<u>normal forms</u>

So far I've only talked about the character of account data. Now I'll consider its structure. I'll start by reviewing the fundamental design concept of normal forms.

Normal forms are conditions that can be applied to test the structure of relational tables, or files as they're known in 4D. The normal forms are hierarchical: a structure that satisfies the one automatically satisfies all of those "beneath" it. There are five normal forms but we'll only be concerned with those up to the third.

A file structure is in third normal form if:

- the information in the key fields is sufficient to uniquely identify each record, and

- all the nonkey information is independent.[1]

Or to quote Fabian Pascal's definition:

> "The objective is to represent 'one fact in one place,' where each column (i.e. field in 4D) represents 'a fact about the (primary) key, the whole key, and nothing but the key.'"[2]

From a practical standpoint you use normal forms as tests to measure how close a structure comes to the ideal. Structures that do not satisfy the various normal forms are not necessarily wrong, but they may suffer certain problems as we will see. We are often forced to stray from the guidelines of the normal forms either because it's easier to do so in the short term, or because it results in certain operations performing more quickly. In the long term, however, non-normalized designs are always more difficult to modify and maintain.

file structure

What does the third normal form mean for storing account information? It tells us that we need to store different kinds of accounts in different files and similar types of accounts in the same files.

From what's been said it's clear that general ledger accounts and subaccounts are dependent because they represent the same history of the same business. On the other hand the way the information in the two kinds of accounts is used is quite different. Their differences would become even more evident if we considered the reports associated them. Even without that detail it is clear that general ledger and subaccounts should be stored in different files.

The role each account plays determines its accounting category. Because general ledger accounts summarize the subaccounts related to them, the category assigned to a general ledger account must be the same as the category assigned to all its related subaccounts.

In accordance with the third normal form's "one fact in one place" rule the category must either be stored with the subaccounts or general ledger accounts. Storing the category with the subaccounts would lead to the case where different subaccounts, necessarily of the same category, all stored the same category information. This represents a dependency between nonkey information, which violates of the normal form condition. The proper place to store the category is in the general ledger file.

Are accounts of different categories sufficiently similar to be placed in the same file?

The accounting category has little or no effect on the kinds of information associated with an account. To a large extent accounts of every category record the same basic information.

The rules of normalization don't say much about how similar information should be in order for it to be stored in the same file. Good design generally opts for simplicity whenever possible, which means avoiding unnecessary files. I subscribe to the principle that data with similar structure should be stored in the same file. This isn't required in theory, but it's a virtual necessity in 4th Dimension considering how 4D displays and prints records. For example the commands **Modify Selection** and **Print Selection**, two of the most powerful commands in the 4D language, only operate on the current selection in one file. Displaying or printing a collection of records from different files takes a lot more work.

From an accounting perspective you certainly will want to print reports that involve accounts from different categories. A balance sheet, for example, draws on general ledger accounts from all categories, It's pretty clear that the best design places accounts of different categories in the same file.

related files

As I mentioned, accounts may contain different kinds of information. To simplify things let's assume that this is only true for subaccounts — certain subaccounts record different information from other subaccounts, and certain subaccounts are related to records in different files. The simplifying assumption that all the general ledger accounts have the same structure is actually pretty good.

The advantage of working with a relational database is that we can store related information in separate files and combine it when needed. In this case we have separate files of clients, suppliers, and various other entities related to subaccounts. The relationship between subaccounts and related files could be one-to-many, many-to-one, or many-to-many.

different types of information

In addition to supporting different file relations, some subaccounts may store different types of information. For example a checking subaccount may track check numbers, a quantity that has no meaning for a tax payable subaccount. The tax payable subaccount, on the other hand, might store an employer ID that's not relevant for checking subaccounts.

The rules of normalization tell us to group our fields into independent sets. These units, called "atomic values," constitute independent logical units and should be stored in their own files. However, normalization does not say that *every* independent quantity deserves its own file.

We are left to decide for ourselves how much independent information to pack into a given file. Is it worth storing a check number counter in a separate file? Probably not, although you could relate the subaccount to a bank record where this counter value would reside.

Just because normalization theory gives no clear direction of how to organize independent information doesn't mean this is a minor issue. In fact it's the major issue underlying object oriented databases, but that's another story. In the file structures that follow I've omitted all the information that is applicable only in certain cases. It's up to you to decide, for example, where to put check sequence and tax identification numbers.

4D file structure

We're ready to draw a realistic 4D file structure for a general accounting application. The following list summarizes the basic conditions this structure must satisfy:

- General ledger accounts and subaccounts are stored in separate files.

- Both general ledger and subaccounts records carry an account balance field.
- The general ledger records carry a category field.
- Multiple subaccounts can be related to a general ledger account (subaccounts have a many-to-one relation to general ledger accounts).
- Every subaccount is related to one general ledger account.
- Every general ledger and subaccount has a name, account number, and ID key field.
- Some subaccounts are related to records in the clients file and some are related to records in the suppliers files.

Notice that I've specified that both the general ledger and the subaccount records carry an account balance. But the general ledger account balance is equal to the sum of its subaccount balances. So this requirement means that the same information is being stored in two places. Since this condition clearly violates the third normal form why do I do it?

This is a good example of where performance requirements win over the guidelines of better design. Performance is an issue here because there are cases where hundreds or thousands of subaccounts may well be related to one general ledger account. To avoid having to perform a huge number of additions every time the user wants to review the general ledger we store a copy of this total with the general ledger record itself.

The trade off is that we risk situations where the stored total and the calculated total disagree. To prevent this we need to exercise special care in updating and maintaining the general ledger. This is typical of the kind of data integrity risk inherent in a structure that doesn't adhere to the normal forms.

Figure II shows a file structure that satisfies the previous conditions. In this case we've hard-coded a link between the subaccounts and clients, and between the subaccounts and suppliers. The relationship is one-to-many because it supports any number of client and supplier records related to one subaccount.

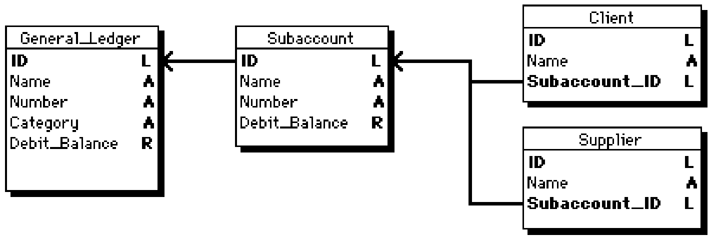

Figure II. One-to-many relation between subaccounts and related files.

An alternative is to store the link to the related file with the subaccount as shown in Figure III. In this case the subaccount stores the file number of the file it's related to and the ID of the related record in that file. I call this a meta-many-to-one structure because it supports many subaccounts being related to one record in any file.

The relationship of records in Figure III is not hard coded because it is not determined by the file structure diagram. The user can determine which record in which file is related to each subaccount. This has a clear advantage in flexibility. However the lack of an explicit link between related files precludes certain automated features in 4D. For example, without an explicit link, 4D Server cannot retrieve selected fields from the related files in certain reports and output layouts.

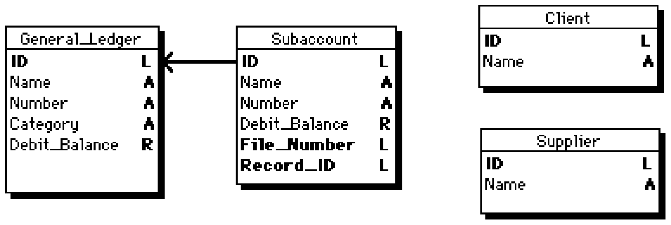

Figure III. A "meta-many-to-one" relation between subaccounts and related files.

The independence of the general ledger and subaccount files from the related information is an important form of modularization. It means that the account information can always be supported by a core file structure, while the business-specific information can be "hooked in" through relations to other files. These file structures both provide a starting point for building an accounts DBMS that's largely independent from the rest of the system.

references

1 An Introduction to Database Systems, volume I, 5th ed., by C.J. Date, Addison-Wesley , 1990. page 533.

2 Understanding Relational Databases, with examples in SQL, by Fabian Pascal, John Wiley & Sons, 1993. page 55.