Accounting Transactions IV: Invoices

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

This is the final installment in a four-article series exploring different types of accounting transactions. In the first article (Dimensions v. 3, n. 6, September / October 1994) we considered the basic logical and relational structure of transactions. In the second article (Dimensions v. 4, n. 2, January / February 1995) we illustrated this using cash transactions as an example. In the third article (Dimensions v. 4, n. 3, March / April 1995) we considered the technique of batch processing in the context of point-of-sale transactions.

In this article I focus on invoices, which are the most complex accounting process where data management, business rules, and accounting requirements all come together. Invoices are handled using an invoice file designed so that it essentially stores no accounting information. Accounting information is stored in related files that have a generic, reusable structure.

The method described here for handling accounting information is patent pending. Its use in commercial applications may be subject to limitiations. Contact Braided Matrix for more information.

Invoices Defined

An invoice records a sale of products to a customer. It lists the customer's name and address, the terms of sale, the products purchased, extra charges, and a total amount due. Before describing the mechanics of invoice operations let's analyze their purpose and use.

The business commits itself to a sale on the stated terms when an order is specified. The customer agrees to accept delivery and pay according to the same terms. The invoice acts as a legal document formalizing this trade agreement.

Invoicing also records accounting information: information that contributes to such items as shareholder profits and taxes payable. These items also have legal liabilities, so the invoice acts as a legal document in this respect as well.

The invoice's accounting contributions may not occur at the same time as the events affecting trade. In fact, they generally occur at different times: the trade obligation begins when the invoice is entered; the accounting obligation begins when the goods are shipped (or delivered).

In complex invoices both the trade and the accounting information continue to accumulate as the invoice passes through its various stages. The computer invoice system must process both trade and accounting information. Each is subject to different rules, each may be handled at a different time, and each needs to be presented to the user through a flexible and ‘friendly’ interface.

Invoicing may also be coupled with the management of inventory. Standard inventory management tasks include:

• Items are committed when ordered, though they remain in stock;

• Items are removed from stock when they’re shipped;

• Older (or newer) items may have to be shipped first;

• Both the price and the cost of the items are recorded.

Invoice data management is so complex that the accounting aspect is actually one of its simpler components. Since the present focus is on accounting, we'll make a series of assumptions and simplifications that will help us to avoid becoming overwhelmed with a plethora of trade, accounting, and inventory rules. We begin by breaking the invoice into three parts:

• Customer;

• Line items; and

• Accounting.

Customer

Each invoice must be related a customer. The customer is the person or company who has agreed to receive and pay for the goods. We'll make the functional assumption here that the customer is selected from a list of customers already on file.

When a customer is specified the system looks up the billing and shipping address stored with each customer, and displays this on the invoice. The user can edit the address information. The addresses will be stored with the invoice. Once the invoice is entered, changes to the addresses stored with the customer will not affect it.

The customer specification area is shown in Figure 1. This includes a list button that opens a window displaying a list of customers. The billing and shipping addresses are displayed in enterable areas. These addresses are reset whenever a new customer is specified.

In this interface areas with half outlines are directly enterable. Those with dotted outlines are not enterable. Nonenterable information is either looked up from related information, or is calculated by the system.

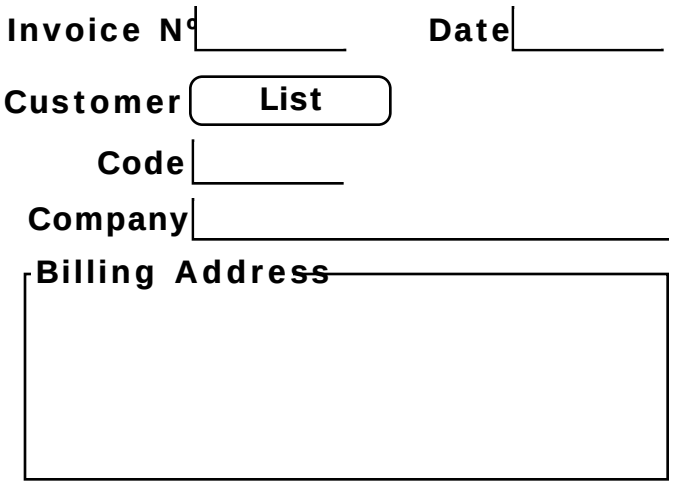

Figure 1: Invoice detail and customer entry user interface.

Line Items

We'll simplify line item entry in the same way that we simplified customer entry. That is, we'll say that the user must select line items from a list of items already on file. To add a new item the user presses an 'Add' button. He or she then selects from a list of inventory items.

We'll avoid the complexity of inventory management by recording the invoice line items without having them affect inventory levels. That is, the ordering or shipping of line items will not change the level of inventory in stock. We'll also say that inventory records store the item's price and cost. It is usually the case that there are several prices depending on the customer or the quantity ordered. In this example the system will only store one price.

In this simplified invoice, when the user selects a new item the system fills in the product's name, code, and price. The user specifies the quantity and can change the unit price. The line under the "Qty" and "$/Unit" headings indicate that the user can enter values directly into these columns. The system computes a total, or extended, price and updates a subtotal. This interface is shown in Figure 2.

Items [Add] [Delete]

Qty	$/Unit	Description	Extension
400.0	0.25	lbs. building stone	100.00
2.0	15.00	bags mortar	30.00
1.0	7.50	trowel	7.50
1.0	250.00	cask of Amontillado	250.00

Subtotal	387.50
Shipping	22.50
TOTAL	410.00

Figure 2: Line item entry user interface.

New records are added to a Line Items file using the Add button, and are displayed in the enterable included layout. The user can change any of the line items by pressing the Modify button. The scripts behind the buttons and entry fields will make the adjustments necessary to maintain consistent totals.

We're not specifying how the system will handle modifications to line items between the time an item is specified and the invoice is ultimately entered. This is an unrealistic simplification of the data entry process. A real application will cache changes in memory until the whole invoice is saved. This can be done using arrays or temporary subrecords (see Dimensions v. 3, n. 1). However, once the invoice is entered the line items are written to their own file.

Our current model can be viewed as a description of the final storage structure. It lacks information about the temporary storage required for the operation of the user interface. Having noted this omission, we will ignore the question of how temporary changes to the line items are stored.

Accounting

The accounting that goes on behind the invoice records the transfer of goods and the obligation to pay for the goods. This could include a combination of cash and credit. It also records the expenses that were incurred in selling. This could include a wide variety of expenses that may be different for every business.

The Revenue Side

To simplify the accounting we'll assume that no cash changes hands at the time of invoicing. The full invoice amount is billed to the customer, and is receivable from the customer once the items are shipped. This means the sale will involve two accounting entries: one records the amount of the sale, the other the amount due from the customer. The first is a credit to a 'sales' account, and the second is a debit to the customer's 'receivable' account. This takes care of the revenue half of the event.

All customers will have their own receivable account. This account is debited every time a customer is charged for goods or services through an invoice. The receivable account accumulates the total amount the customer owes to date. This is given by the account's total debit balance. When the customer provides payment, a negative debit amount is entered. A negative debit is equivalent to a credit.

The credit, or negative debit, lowers the total debit. This reflects the fact that the customer now owes less. The entry of customer payments is handled through some other entry screen; that is to say, it is not handled through the invoice. It is not considered further.

A single sales account is credited by every invoice. This is the user's general sales account, whose accumulated credit gives the total sold on all invoices to date. The sales account is a default account that must exist in the system in order for any invoices to be entered. This contrasts with customer receivable accounts that are linked to customers. These are only created when new customers are entered.

If you think about what is happening in the sale process you will realize that this revenue information does not tell us the profit. To know the profit you have to know both the revenue and the expense. In this case the expense is the cost of the items being sold.

The Expense Side

Both the cost of each item and the price of each item are stored with the item's inventory record. Unlike price, however, the user cannot change the cost. The cost is predetermined and is not affected by the choice of customer or terms. Along with accounting entries that record the amount of the sale, there are additional entries that record the cost of the sale. Since the costs can be determined as soon as the user specifies the items and quantities, the system can handle the expense entries automatically, without the user's intervention.

What are the accounting rules for handling cost? Double entry accounting uses two accounts for this purpose. One is the inventory asset account, which

we will call the ‘asset’ account for short. The other is called the ‘cost of goods sold’ account.

The asset account records the value of inventory items in terms of their acquisition cost. These costs are entered when items are acquired, and removed as items are sold. The balance in the inventory account tells you the total cost of what’s available.

The cost of goods account accumulates the same costs, but it accumulates them as the items are individually sold. The balance in the cost of goods account tells you what it cost to acquire those items that have been sold. The process of moving assets through the asset account to the cost of goods sold account is shown in Figure 3.

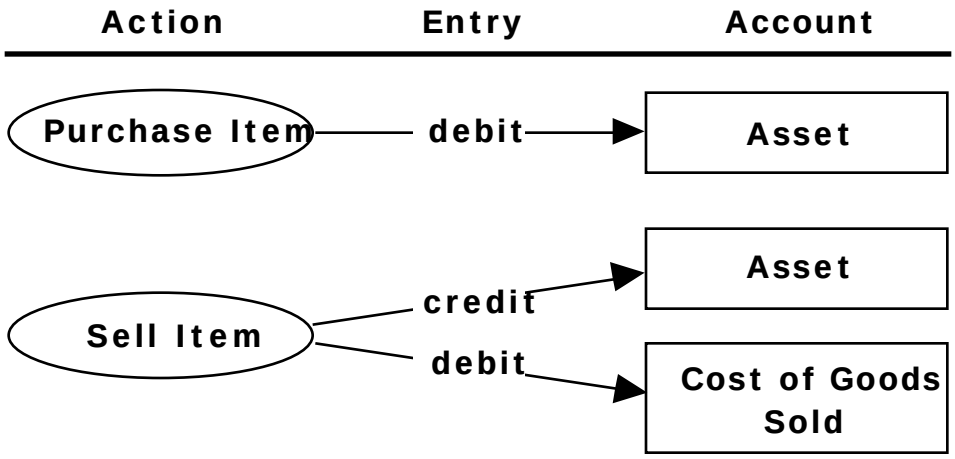

Figure 3: Flow of information through the inventory accounts.

The expense entry that’s made through the invoice credits the inventory asset account and debits the cost of goods account. The amount of these entries is determined solely by the cost of the items ordered.

The fact that the asset account entry is always recorded as a debit, and the cost of goods sold entry is always a credit, is specified by general accounting rules. These rules say that the depletion of the asset account, any asset account, is recorded by crediting the account.

If a lowering in value being signaled by a credit is confusing to you, the source of your confusion lies in mixing up the colloquial and accounting definitions of credit: they are not the same. If you are involved in accounting, then the sooner you forget the colloquial definition, the better.

To summarize, when entering an invoice the system debits a receivable account, credits a sales account, debits an asset account, and credits a cost of goods account. An example is shown in Table 1 below. These four entries are always made, although in a real system other accounting entries would be made as well.

The values and the accounts change according to the items and customer specified. The user may fiddle with the sales price, but the expenses are fixed and are handled by the system automatically.

Account	Debit	Credit
Customer Receivable	410.00	
Sales		410.00
Cost of Goods Sold	210.00	
Inventory Asset		210.00

Table 1: Invoice debits and credits.

Storing the Information

Requirements

The invoice has external and internal information requirements. External information is that obtained through user input. This includes specifying a customer, address, date, and line items, along with items, quantities, and prices.

Internal information consists of the data needed to support the relational file structure. It is comprised of the primary and foreign key values that link

records between the various files. Internal information is handled programmatically and is invisible to the user.

Figure 4 shows the flow of information in the invoicing process. The ovals label data stores. These are separate entities from the user's point of view. They need not be files in the relational sense. For example, comparing Figures 4 and 5, you can see that the invoice store represents information stored in the invoice and line items files. The transaction store represents the transaction and components files.

The arrows in Figure 4 show that the user supplies invoice information in three ways: directly, with reference to the customer store, and with reference to the inventory store.

The invoice itself then assembles an accounting transaction that represents the revenues and expenses associated with the invoice. The system then refers to the components, and updates the balances in the related accounts. The invoice transaction is automatically created, and the account balance automatically updated, when the user saves the invoice.

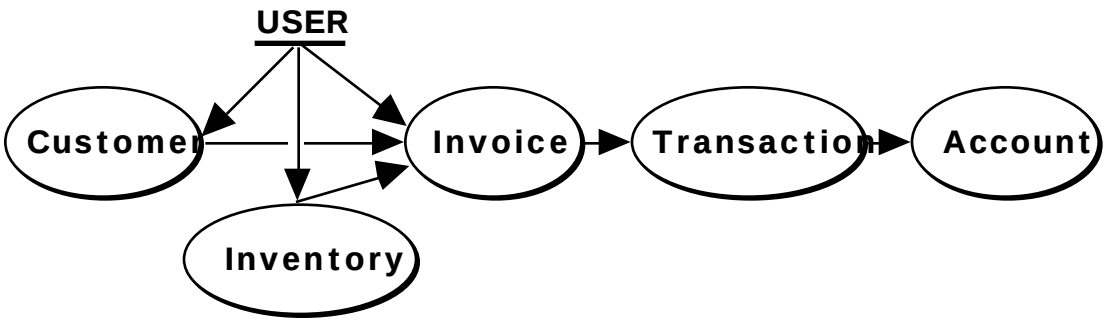

Figure 4: The Data Flow Diagram of the information playing a role in the creation of an invoice.

The Related File Method

The central principle of relational database design is to store independent but related information in distinct, related files. We can apply this concept to accounting systems by separating accounting data from data that are specific to the invoice's other business functions.

The accounting data generated by an invoice consist of debits and credits that record the transfer of assets from one account to another. From the accounting perspective everything the invoice does can be reduced to answers to the following questions:

- How much has been sold?
- How much has been paid?
- How much is still due?
- What was the original cost of the items sold?

Any level of detail can be recorded in terms of a transfer of assets between accounts. The particular accounts involved will depend on what the invoice actually does.

The non-accounting information recorded through the invoice specifies the agreement between the buyer and the seller. This information is given by the answers to the questions:

- Who are the parties involved?
- What has already been delivered, and what has been promised?
- How will outstanding items be delivered?
- What are the terms of payment?

Answers to these questions will differ for each business. As a result, a general invoice file structure — one applicable to all businesses — may not be possible. This contrasts with the accounting file structure, which is applicable to all business situations.

The essential point is that:

If a solution for storing invoice data can be found that consists of two, relatively independent parts, one of which is generally applicable, and the other fitted to the specific circumstance, then the general parts can always be reused in any new invoicing application.

This reusability provides advantages in the following three situations:

• New software is needed for a new business application.

• A new type of invoice (or invoice-like element) needs to be added to an existing application.

• The invoice in an existing application needs to be modified to a significant degree.

In each case, the accounting structures and functions are already in place and only the business-specific areas will change. Where changes are made to an existing application, the new information structures can be added without disturbing the existing accounting information. Changes in the invoice file and data structure will not affect the structure of the accounting information. How this is accomplished, and exactly what changes and what does not, will become clear when we consider the actual file structures involved.

File Structure

Relational structures usually involve items that stand in a many-to-one (or one-to-many) relationship to each other. The structure used here centers around a one-to-one relation — splitting a single invoice entry into accounting and business-specific parts.

The information on the accounting side is stored in a transaction record, whose components are related to the transaction and to numerous account records. The information on the business side is stored in an invoice along with line items related to the invoice and to inventory records.

This decomposition is supported by two one-to-many relationships: the transaction and its many components, plus the invoice and its many line items. It is the invoice and the transaction that exist in a one-to-one relation.

Each accounting transaction consists of debits and credits. Each debit or credit affects a particular account, and the component records that are linked to the transaction record reflect these entries.

Business rules governing the invoice process determine which accounts are involved in the accounting transaction. In this case these are references to a

general sales account and a cost of goods sold account. These references are hard-coded into the code that supports the invoice process.

The identity of the receivables account is determined by the selected customer. The inventory items listed on the invoice determine which asset accounts are involved.

This information does not have to come from these sources, nor be limited to these accounts. Different accounts can be used for different types of customers or different terms of sale. The only consequence will be different account and component information, but it will still be supported by the same files. This is exactly the attraction of this modularization of the structure.

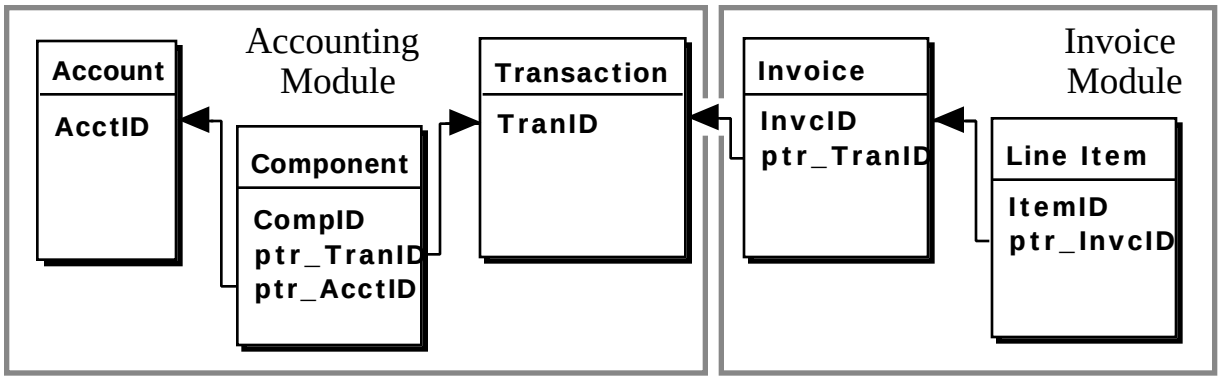

Figure #5: Relationship of the files used to store invoice information.

Analysis of the Method

Modularity offers many benefits, but it does not come without its costs. Power and flexibility are balanced by greater complexity and tougher design requirements. Modular systems are never the simplest: they are suited only to development projects that aim to produce applications with the greatest long-term value.

The Benefits

The invoice structure can be modified without impacting accounting. This means the structure is able to incorporate changes in business operations without requiring much change in accounting. For example, variable item

pricing and various payment terms could be added without affecting accounting. Sales commissions, license fees, discounts, taxes, and interest charges could similarly be included with only minor changes.

• Batch processing is implemented more easily:

The invoice can store all the information needed to create the accounting entries. The entry of the accounting information can be done when the invoice is entered or it can be deferred until a later time. Batching is done to minimize the problems of network traffic and data access, and to speed data entry. I discussed other benefits and drawbacks of batch processing in the third article of this series (Dimensions v. 4, n. 3).

• Easier to maintain:

Since the files and the functions required to handling invoice and accounting information are largely encapsulated, changes in one area are less likely to affect operations in the other. This means that these areas can be modified, extended, tested, and debugged relatively independently. Modularity also means that a programmer can master one area without having to learn other areas at the same time.

• Modules can be reused:

The accounting files and procedures, if crafted generically, can be copied into other applications. They can also be used in other contexts, such as for the processing of purchase orders, payroll, inventory or project management. Reusable code can save large amounts of development time and produce applications of higher quality. In addition, applications built on reusable components inherit all the virtues of modularity and maintainability.

The Drawbacks

The main costs of modularity are greater complexity and the necessity of more careful design. The complexity comes from the following quarters:

• Complex read / write access:

It takes more work to gain read / write access to files in a modular system because more files are related. Complex relational structures require more

code to manage record access and to save related data. This requires advanced techniques (see Dimensions v. 2, n. 2 & 3).

• Widely distributed information:

It initially requires more work disassembling and reassembling information stored in the related files. While it is simpler and quicker to use fewer files, this generally results in rigid and specific application. If this is not the objective, then a more carefully designed application will likely be less work in the long-run.

• Sophisticated Design Requirements:

Reusable components must be carefully designed. They must be relatively simple in order to be comprehensible and manageable. They must be both general and flexible if they are to find a use in other contexts.

If you had the foresight of knowing how your objects would be reused, then it would be much easier to design them. But generally you can only guess how they will be used in the future. Developing skills to design with few long-term specs requires learning the more difficult techniques of object-oriented analysis and design.

Overview

We've described a general method of decomposing invoice information into accounting and business-specific components. These components reside in files whose relations reflect the rules of accounting. Because the file structure is independent of the rules of business operations, it applies to any invoicing situation.

This file structure is based on modularity and encapsulation. It is suited to software projects that aim to maximize the long-term value of the applications they produce. These ideas come from the field of object-oriented analysis and design. While I personally find this field the most exciting area of software design, it is not for all programmers or all projects.