Accounting Transactions III: Batch Processing

By Lincoln Stoller, Ph.D.

This is the third in a four-article series exploring different types of accounting transactions. In the first article, “Accounting Transactions, I,” Dimensions v.3, n.6, September/October 1994, we considered the basic logical and relational structure of transactions. In the second article, “Accounting transactions” Dimensions v.4, n.2, January /February 1995, we used cash transactions as a concrete example. In this article we focus on the technique of batch processing, which we develop in the context of point-of-sale (POS) transactions.

Batching Defined:

Batching is a general technique in which data processing is done in a series of stages. Information is only partially processed at the time it’s entered and it’s recovered at a later time in order to complete the processing. We’ll call the first part of the process “batch preparation” as it involves entering data and partially updating the rest of the database. The second part of the process, which we’ll call batch processing, involves recovering the prepared items and completing the datafile updating process.

In this example POS transactions create new records in a transaction and a component file and also update balances in an account file (see figure 2). Batch preparation consists of storing data in the transaction records but not updating the component or account records. When the batch is processed we will update all the related files for many POS records at once.

There are many familiar examples of staged data processing. For instance your accounts for on-line services are updated every few days to reflect past usage. United Parcel Service uses hand held computers to log delivery information which is downloaded to a central system and batch processed.

Batch processing offers several benefits and suffers several drawbacks. On the positive side, data entry is faster and batch processing allows an extra degree of control. On the negative side it is more complex and requires more training both to manage and to program. We’ll consider each of these issues in detail

The Benefits:

Partially entering data is faster than fully processing the data because it defers several steps. These steps need to be done later, and the burden of batch management adds additional processing. Often, the total amount of time required to prepare and process entries is greater than if all processing is done in total for each entry. But what's important is that data entry appears to be faster because the user regains the system's attention after a shorter delay.

Batch processing adds extra control by enabling the user or administrator to modify, or add information to the batched entries before their final processing. The user doesn't have to be as concerned with accuracy or clarity and can focus instead on dealing with the immediate situation.

For a specific example consider the process of preventing duplicate entries. This is a data intensive task that usually requires the review of many entries. While you could check a given entry against all possible entries every time a new entry is made, but there are other techniques you can use for a group of entries that can speed the process considerably. In this case a batch processing strategy would shorten total processing time.

Finally, by selecting which processes to defer and which processes to perform immediately, the programmer can prevent bottlenecks that could slow or even prevent entries. In the examples we've dealt with in previous articles each transaction updates an account balance field stored in an account record. When many users need access to the same account each user will have to either wait their turn for record access or cancel their entry.

Any situation where multiple users access few records is a potential bottleneck. The topic of handling bottlenecks is a discussion of its own so let it suffice to say that bottlenecks can both slow the system and place it at a greater risk failure. Batch processing can completely defer access to centrally used records at the batch preparation stage.

Final processing can focuses on updating a centrally used records. The update process funnels all information through this bottleneck, but because the batch is handled by the programmer the information can be screened, summed, or otherwise prepared to minimize contention and repetitive actions. In addition, the batch can be processed at a time when network traffic and data contention is low.

Avoiding bottlenecks is probably the most compelling reason to employ batch processing. Dividing data entry into a user entry phase, and a carefully controlled update process relieves the system

designer of many speed limitations and most of the difficulty record of contention.

The Drawbacks:

The two main drawbacks of batch processing are the delay in gaining full access to the information and the extra programming that's required. There are also less immediate drawbacks that arise when the batch processing is employed to work around system limitations. Batch processing is not a simple process and while it can resolve some hardware limitations it also imposes its own burdens. Even if the process is designed so as not to hinder users, it may do so later when system specifications change.

I remember 20 years ago, at the University of California, when users did have not direct access to central processors. Programs were read into card readers and queued to run in a batch. Depending on the length of the program and the priority of the account your program would be run sometime between ten minutes and ten hours later. I'm sure MIS was perfectly happy — but in retrospect these were dark days for computer users.

It is easy for system designers to forget that a successful design is determined by how well it helps the users, and not by how well it works judged on engineering criteria. Systems are becoming more interactive and data more immediately available. Batch processing runs counter to these trends.

Point of Sale Transactions:

A point-of-sale transaction is a transaction entered at the time a sale is made and which must perform according to the user's time constraints. By "user's time constraints" we mean the time-frame of the sales event, which may be less than the time the program requires to enter and update all the data. If the user needs to regain control immediately, batch processing offers a solution.

We'll consider a simple cash sale involving two accounts, a Sales account, and a Cash account. The amount of the sale is credited to the Sales account and debited to the Cash account. There will be different sales accounts to track different types of sales, which the user specifies at the point of sale.

Analysis of the Batch Process

Batch processing involves separating data entry into batch preparation and batch processing. The first involves entering data

and preliminary storage, the second involves bringing the data file fully up to date.

The data flow diagram (DFD) in Figure 1 shows what occurs done when a user enters a POS transaction. Processes are represented by ovals, data stores are represented by rectangles. Lines coming from rectangles to ovals represent the user drawing information from the data stores. Lines from the oval to the rectangles represent the process of storing information.

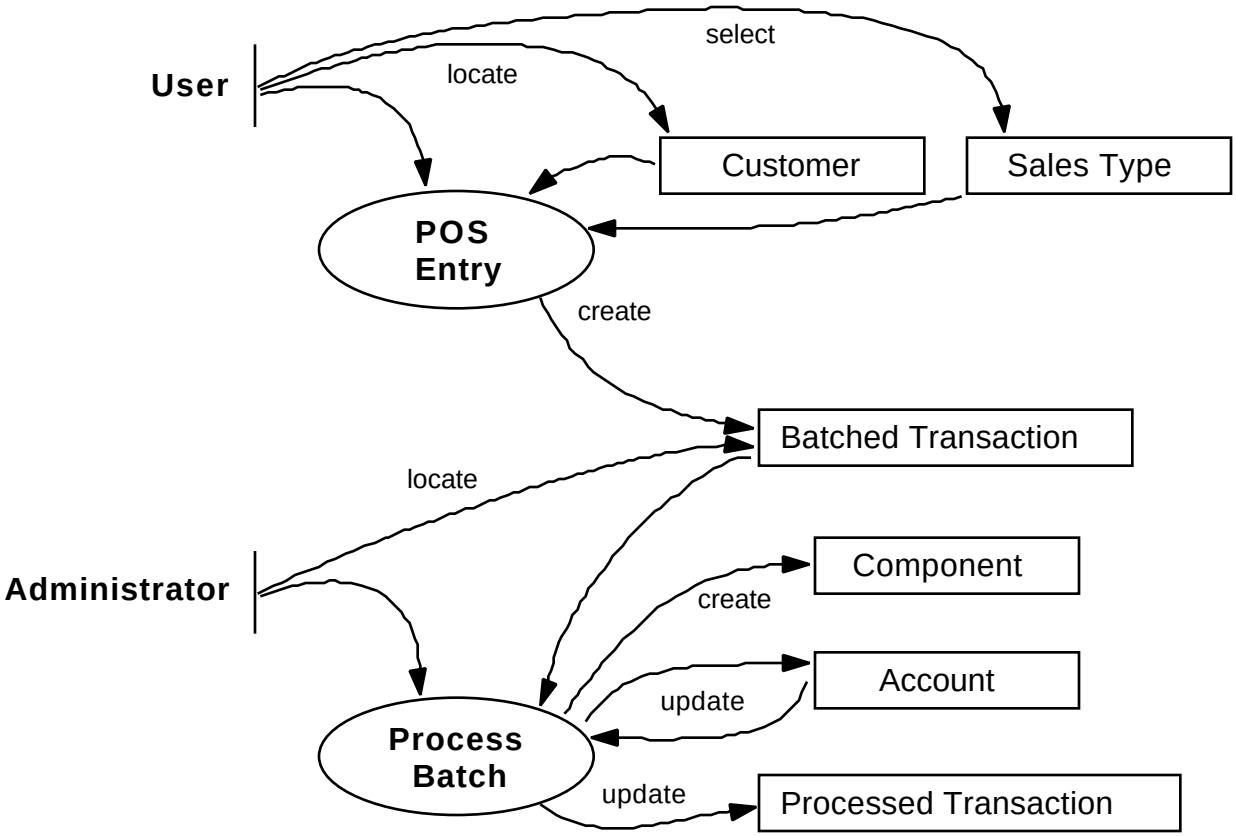

Figure 1: Batched POS Transaction Data Flow Diagrams.

The top portion of figure 1 shows the preparation stage. The user specifies a date, customer, sales category, and sales amount. The customer must correspond to a record in the Customer file. The sales category is chosen from a list of mandatory categories. A minimal amount of file updating takes place at this stage. In particular, component records and account records are not modified.

The bottom of figure 1 shows the final processing. Here we return to the partially processed records, read the temporary information

stored with them, and update the related records in the Component and the Account files.

Structure Chart

One of the consequences of splitting POS into two parts is the need to support two interfaces: one for data entry (batch preparation) and another for batch processing. A blueprint for these areas is shown in the following System Structure Chart (figure 2).

Figure 2 shows two distinct interface areas. The data entry area, on top, is controlled through an input layout. This layout supports the look-up of customer and type-of-sale information, and offers the usual option of accepting or canceling the entry.

The batch processing area, shown below, uses an output layout to display current sales entries. The output screen is the appropriate place to handle batch processing since this is an action that applies to a selection of sales records. This output screen supports the usual add, modify, delete, search, sort, and subset selection options.

The output layout also supports a menu labeled “Batch Processing” that allows the user to locate records queued for processing as well as to initiate processing. Batch processing is placed in a menu so that it can be protected with a password.

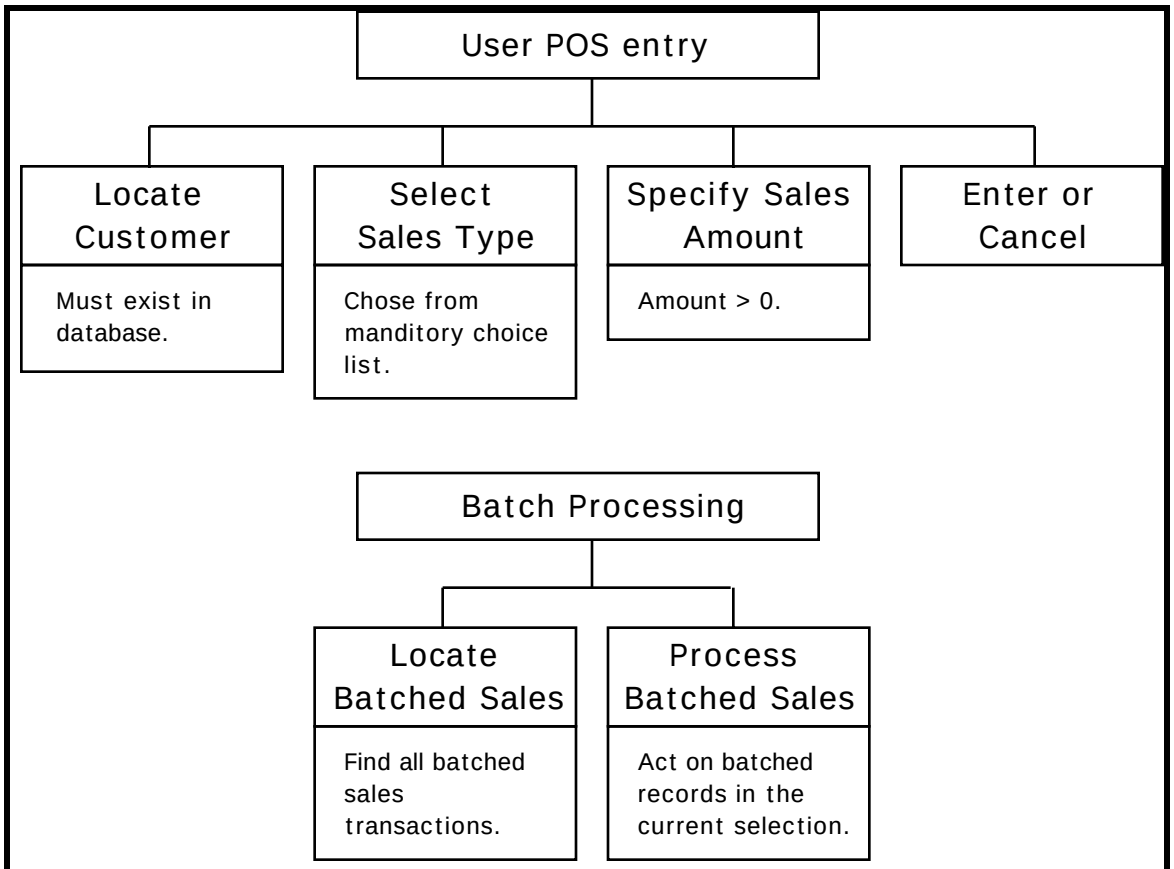

Figure 2. Data Entry and Batch Processing Structure Chart.

Information Requirements

POS entries store what we'll refer to as external and internal information. External information has to with the world outside the database and includes all the information about the actual POS event and how it's to be handled. Internal information consists of the data required to support the relational structure and performance requirements. These include primary or foreign keys, derived and denormalized information, flag values and compound keys.

External Data Dictionary:

The External Data Dictionary (EDD), in table 1, lists the information that the users enters and which reflect the user's understanding. It also includes the business rules that govern this data.

Table 1: External Data Dictionary

Item	Field Type	Enter-able	Description
Date	date	yes	Initialized to current date.
Sales amount	real	yes	Must be positive.
Customer name	alpha	yes	Looked up from the customer file.
Type of sale indicator	alpha	yes	Selected from popup menu, used to determine the value assigned to the Subtype field.

File Structure and the Internal Data Dictionary

The articles in this series all make use of a structure where transaction information is stored in a Transaction file that is in a one-to-many relation with a Component file. Each transaction has a single transaction record and multiple component records.

The whole file structure consists of a Transaction file (which stores information common to all components), a Components file, a subfile used for temporary storage (attached to the Transaction file), and the accounts file. This is shown in figure 3.

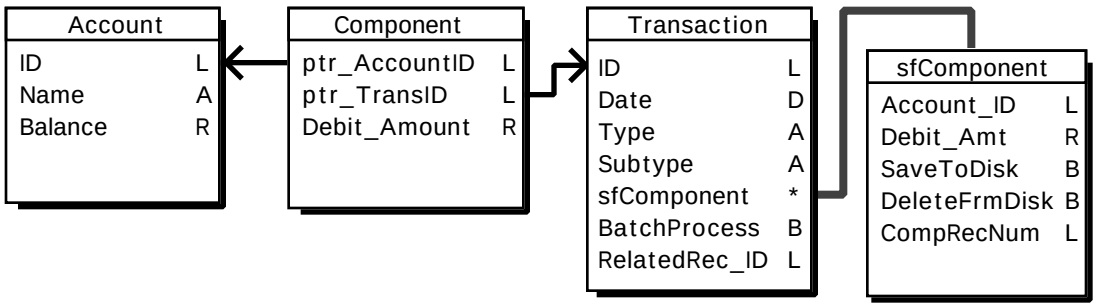

Figure 3: 4D transaction and accounts file structure.

We can easily use this structure to support batch processing by storing the initial POS information in the Transaction file and delaying the creation of components and the updating of accounts until a later time. We will also add a few extra fields to the transaction file to better support batch processing. Referring to the file structure of Figure 3 and the functions listed in table 1, we can complete the internal data dictionary that is given in table 2.

Table 2: Internal Data Dictionary

Item	Field Type	Enter-able	Description
Transaction ID	longint	no	Unique and required.
Transaction type	alpha	no	Set to "POS" for entries created through this layout.
Subtype	alpha	no	Set to "BAR_", "REST", or "CATR".
Related record ID	longint	no	Set to selected customer ID.
BatchProcess	boolean	no	Set to True for all new records.
Balancing account information:			
Account ID	longint	no	Assigned according to user selection.
Save to disk	boolean	no	Used to handle record modifications.
Delete from disk	boolean	no	Used to handle record modifications.
Component record number	longint	no	Read from existing components when they exist. Set to -1 if they don't exist. Used to handle record modifications which are not discussed here.

Interface:

POS Transaction List

Different types of transactions are distinguished by the values assigned to their transaction Type field. POS transactions can be listed separately from other transactions by displaying only those transactions assigned a POS type. The Transaction file output layout is specially designed for POS-type records. We can draw information from the related component records, or subrecords in the case of batched records, to display the transaction amount. The output layout is shown in figure 4.

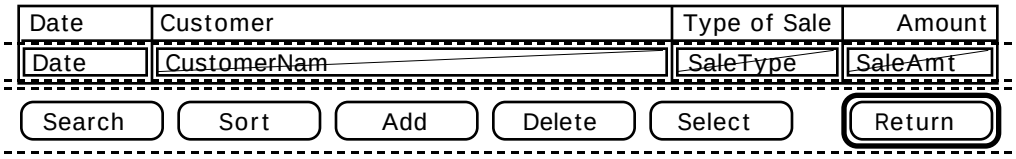

Figure 4: Transaction file output layout for POS transactions.

The functions listed in the structure chart (figure 2) are handled through buttons in the footer area or through the Batch Processing menu associated with this layout and are shown in figure 5. The other functions listed in the footer area are included for completeness but won't be discussed.

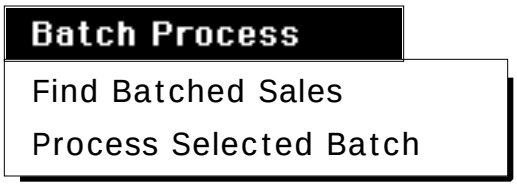

Figure 5: Menu associated with the POS transactions layout.

The two items on the Batch Processing menu allow the user to view the records queued for batch processing, and to process the batched records in the current selection. By reducing the current selection to any subset the user is able to complete processing for any particular batched entries.

POS Input Layout

POS entry is done through a input layout stored in the transaction file. The specifications require that the layout support:

- look-up of customer by name,
- a list of required sales types that the user can choose from,
- a preprogrammed knowledge of which accounts to use for each sales type.

The issue of which accounts to use for each sales type deserves some explanation. Because we're doing double-entry bookkeeping we always log a debit and a credit for each transaction. The credit is to one of the sales accounts and, in this simplified example, the debit is to a single cash account.

In this example we're going to handle restaurant sales. Here a sale of liqueur credits a bar-related sales account, food purchased in the restaurant credits a restaurant account, and off-site food service credits a catering account. Each alternative involves a different sales account and is tagged with a one of the 4 character sales codes shown in table 3.

Table 3: Accounts associated with each of type of sale.

Sales Code	Debit Account	Credit Account
"BAR_"	Cash	Bar Sales
"REST"	Cash	Restaurant Sales
"CATR"	Cash	Catering Sales

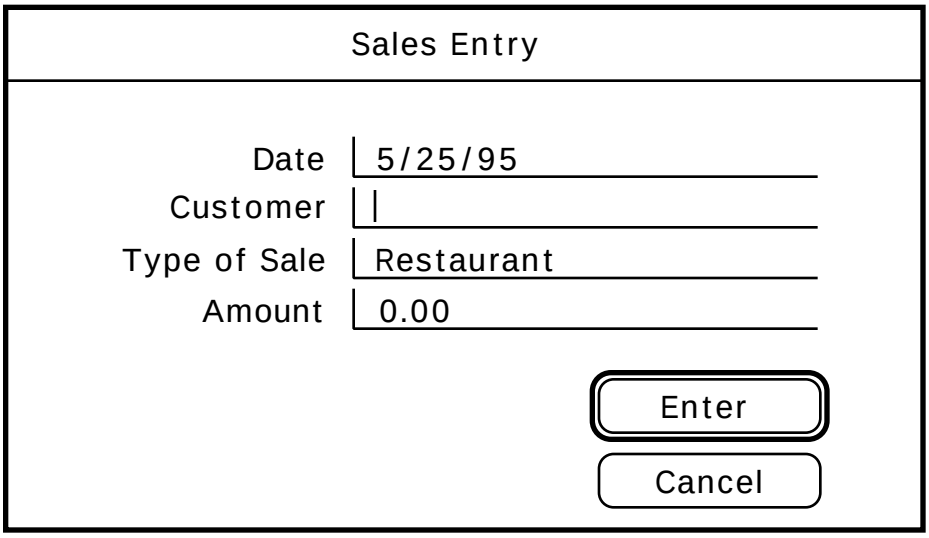

Figure 6: POS input layout.

The input layout shown in figure 6 has four enterable fields. The name field has a script that checks the customer's name against customers stored in the database, the sales type is handled using a choice list, and the sales amount screens data so that only positive values can be entered.

When the user presses the **Enter** button the layout performs the following functions:

• It locates and assigns a transaction record ID number. This number is stored in a separate counter record that needs to be incremented every time a transaction ID is assigned.

• It stores the customer's ID in a field in the transaction record.

• It stores the entry date and sets the "BatchProcess" field to true.

• It considers the type of sale and creates two subrecords. One stores the ID of the account to be debited (the cash account in this case) and the debit amount while the other stores the ID of the credit account and credit amount.

Notice that subrecords are only being used for temporary data storage. Once the batch is processed the subrecord information will be copied to other files and the subrecords will be deleted. This is important since subrecords are generally a poor method for permanent data storage.

Batch Processing Procedure

The transaction file contains records that have been fully processed and records that are awaiting batch processing. Fully processed sales transactions are linked to two records in the components file which, in turn, each relate to an account. Batched transactions use subrecords to store temporary information, fully processed transactions have no subrecords.

Processing batched records is straight forward and involves the following steps.

• Locate the records in the current selection for which BatchProcess = True .

• Loop through the selection creating a component record for each subrecord and copying subrecord information to it. At the same time update the debit balance stored with each related account.

The sample code of figure 7 shows one way to implement this loop. Notice that related account records must be unlocked in order for their balances to be updated. If an account is locked then the transaction it's related to should not be processed.

The loop is executed after starting a 4th Dimensions transaction using the Start Transaction command. The simplest approach would be to blindly update related files based on the rational that the whole 4D transaction could be canceled if a locked account is encountered.

That would be a poor algorithm because a significant amount of processing time could be lost due to a single locked record. We use a more intelligent algorithm in which all the accounts associated with each transaction are placed in a read/write state before making and modifications. As a result we never make any changes that might need to be rolled back and, as long as all referenced accounts can be located, we can always validate the transaction.

```
`Procedure ProcessBatch
`Processes all batched sales transactions in the current transaction selection.
`Uses a 4D transaction that never needs to be rolled back, under normal circumstances.
c_Longint  ($j;$k;$NumBatched;$NumDone;$NumNotDone;$NumComps)
$NumDone:=0
$Error:="None"
Search  ([Transaction];[Transaction]BatchProcess =True ;*)
Search  ([Transaction]; & [Transaction]Type ="SALE")
$NumBatched:=Records in selection      ([Transaction])
If ($NumBatched>0)
    Confirm  ("Ready to process "+String($NumBatched)+ " batched transactions.")
    If (OK=1)
        Start Transaction
        For ($j;1;$NumBatched)
            Search  ([Account];[Account]ID=[Transaction]sfComponent'Account_ID;*)
            Apply to Subselection      (Search([Account];
                                | [Account]ID=[Transaction]sfComponent'Account_ID;*))
            Search  ([Account])
            Apply to Selection     (Account];[Account]Name:=[Account]Name)
            If (Records in set    ("LockedSet")=0)
            $NumComps:=Records in Subselection      ([Transaction]sfComponent)
                For ($k;1;$NumComps)
                    Create Record    ([Component])
                    [Component]ref_TransID:=[Transaction]Trans_ID
                    [Component]ref_AcctID:=[Transaction]sfComponent'Account_ID
                    [Component]DebitAmount:=[Transaction]sfComponent'DebitAmount
                    Save record   ([Component])
                    Search  ([Account]Account_ID=[Component]ref_AcctID)
                    If (Records in selection      ([Account])>0)
```

```
                    [Account]DebitBalance:=[Account]DebitBalance
                                            + [Component]DebitAmount
                Else        `Uh oh, big problem, stay calm.
                    $Error:="Account with ID = " +String ([Component]ref_AcctID)
                        +" is not in the database. Sales transaction with ID = "
                        +String ([Transaction]Trans_ID)
                        +" needs to be corrected."
                    $k:=$k+$NumComps `Exit the inner loop.
                    $j:=$j+$NumBatched      `Exit the outer loop.
                End if
                Save record ([Account])
                Next Subrecord ([Transaction]sfComponent)
            End for
            $NumDone:=$NumDone+1
        End if
        Next Record ([Transaction])
    End for
    If ($Error="None")
        Validate Transaction
        $NumNotDone:=$NumBatched-$NumDone
        $SayLocked:= Num ($NumNotDone>0)*(String ($NumNotDone)
            +" sales couldn't be processed because related accounts were in use.")
        Alert ("If I've counted correctly "+String ($NumDone)
            +" batched transactions have been processed. "+$SayLocked)
    Else
        Cancel Transaction
        Alert ($Error)
    End if
End if
Else
    Alert ("There are no batched sales transactions in the selection, you silly human!")
End if
```

Figure 7: the batch processing procedure.

This algorithm makes use of the fact that any record modified and saved while a 4D transaction is in process remains locked to all other users until the transaction is completed. We combine this with the

fact that the Apply to Selection command saves all the records in the selection to which it is applied.

Notice the Apply to Selection located in the middle of this procedure. This command does not change any value because it assigns the account name to itself, but still saves each record. If any of the records in the selection are locked they are placed in the 4D system set "LockedSet," which we test. If there are locked accounts we skip directly to the next sales transaction. If there are no records in "LockedSet" we are assured that the Account records will remain unlocked for the duration of the transaction. This is important since it enables us to proceed into the inner loop without further concern about locked accounts.

We also test for the unlikely event that a referenced account has been deleted since the batch was prepared. This should be prevented by that part of the database that handles accounts, but should a referenced account be absent we perform a test that will prevent the entry of a one-sided transaction. If a referenced account can not be located the whole 4D transaction is canceled and the user is informed of the problem.

Once the batch update is complete, assuming there are no references to nonexistent accounts, we issue the Validate Transaction command to save our changes to disk. We then tell the user how many items were processed.

One small but significant point is that we don't solicit any response from the user until after the 4D transaction is complete. This way a potentially large set of locked accounts are unlocked as soon as possible.

Overview

In this article we've examined POS accounting transactions and batch processing. POS transactions are a good candidate for batching because they require minimal delay and because administrators may want to review POS entries before final processing. That is, batch processing serves both the end-user and the administrator.

Batch processing is not limited to accounting. It is a general technique useful anywhere data entry can be broken into two parts. It is an often attractive, though rarely optimal, technique for improving system performance. It is best used sparingly and only when the logic of the situation supports a staged processing approach.