Double Entry Accounting in a Relational Database

Robert Chanphakeo · Follow
12 min read · Dec 15, 2016

This entry was taken from web archive — in an effort to preserver it from being lost forever — Author: Michael Wigley

INTRODUCTION

Many computer systems utilising a relational database contain some sort of accounting information. In the initial design and development stages there is often a debate about what type of accounting strategy should be used by the software to record this information. The decision is usually between a simple and cheap ‘single entry’ accounting system and a more complex ‘double entry’ accounting system.

A ‘single entry’ system is one where the numerical values being stored are only recorded once. In a ‘double entry’ system each value is stored twice, once as a credit (a positive value), once as a debit (a negative value). There are a number of rules that control these values. These rules would be recognised by any trained accountant although they may not understand how it is stored in a relational database. The main rules are as follows:

1. Every entry into the system must balance — i.e. the sum of any transaction must be zero.
2. The sum of all the values in the system at any one time must be zero (the ‘trial

balance').

3. No values can ever be amended or deleted. They must be negated with an opposing entry (a 'contra') and re-entered ('re-booked'). This provides a completely secure audit trail.

APPLICABILITY OF DOUBLE ENTRY ACCOUNTING

At the outset of a project the cost of a simple accounting system is always appealing. The complexity and effort of a full blown 'double entry' system often seems difficult to justify. In reality going for a simple 'single entry' system is usually a false economy. If the accounting information is just mirroring a paper system outside the database then a single entry system might be acceptable, however, if any of the following apply then a full 'double entry' mechanism ought to be implemented from the outset:

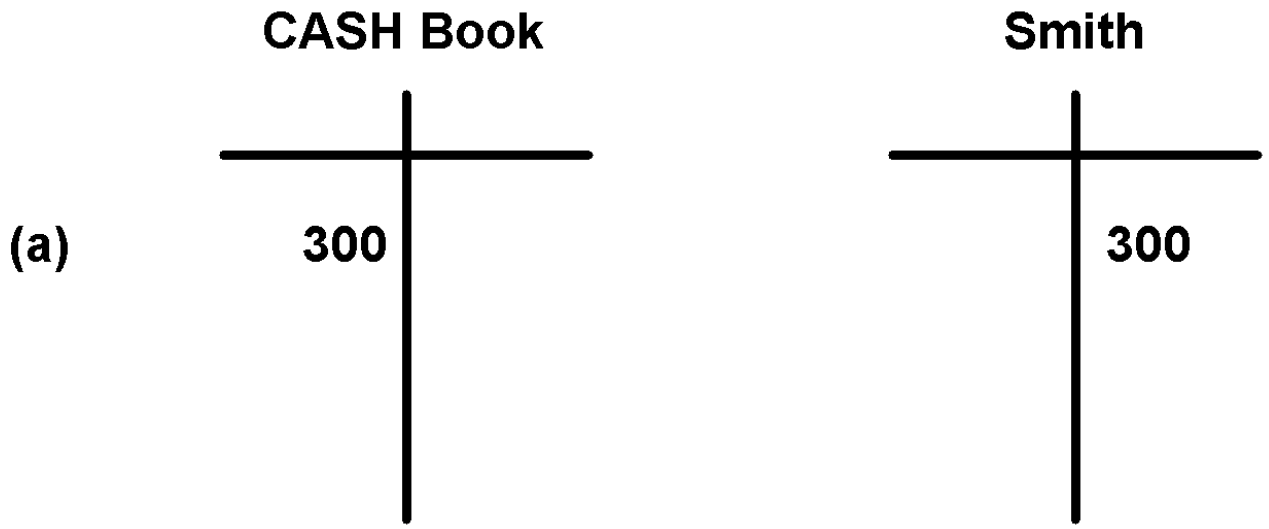
1. If the information will ever need to be audited by an accountant
2. If the information held is the sole record of ownership
3. If the information held represents high value assets.
4. If the system is likely to expand and grow in the future.

A DOUBLE ENTRY EXAMPLE

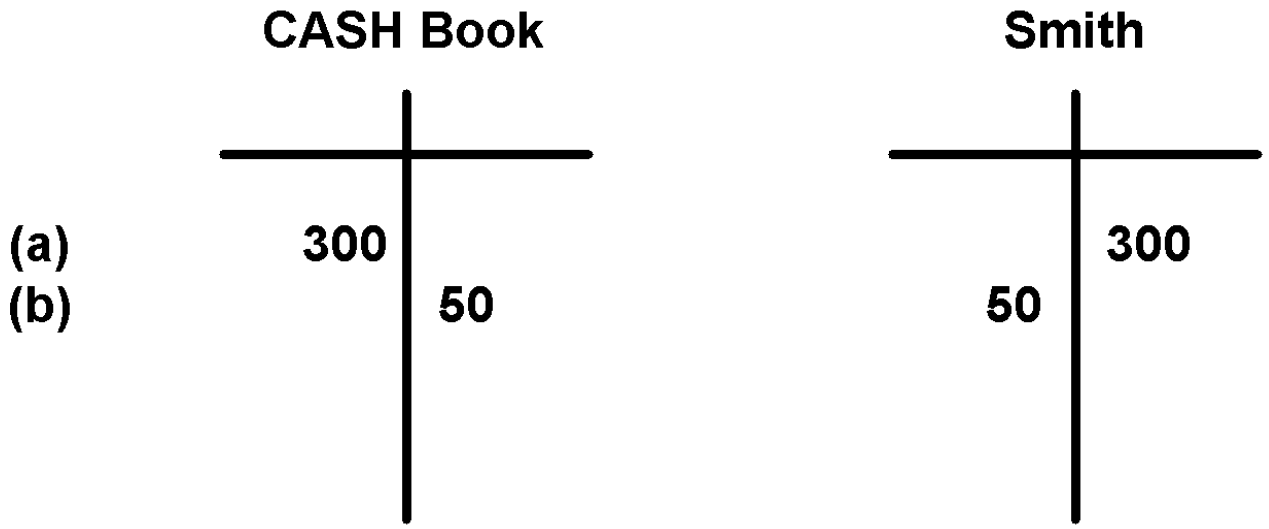
The key factor of a double entry system is the presence of a 'cash book' account. This account contains the entries made when assets (e.g. money) are taken into or released from the accounting system. As such the total value of this account always matches the total value of the assets in the system.

Using accountant's 'T charts' to represent this we use the following example that uses two accounts only. The 'cash book' and an account in the name of 'Smith'.

(a) £300 is entered into this system to be allocated to the Smith account. The £300 is credited to the Smith account (credits are on the right, debits on the left). To match this a debit of £300 is allocated to the Cash Book.

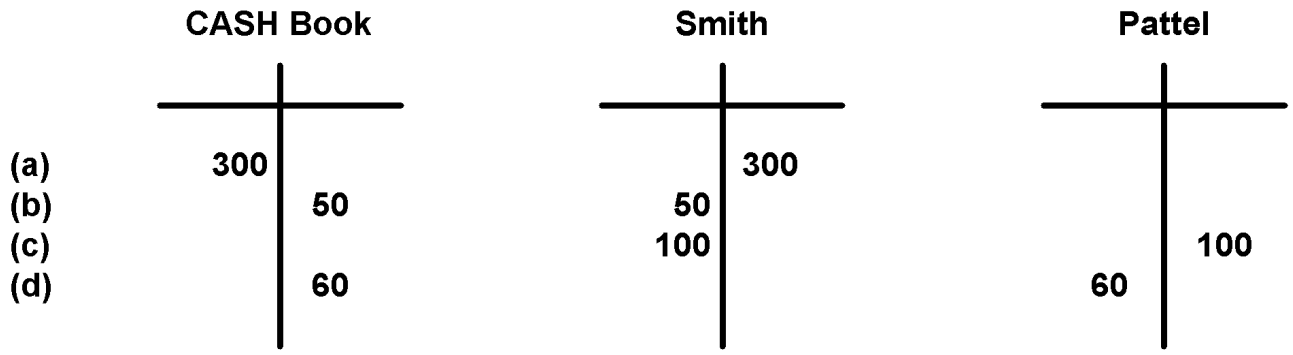


(b) If £50 is taken out of the system from the Smith account, £50 is debited from the Smith account and credited to the Cash Book.



(c) If another account is added in the name of Pattel and £100 is transferred from Smith to Pattel then £100 is debited from Smith and £100 credited to Pattel.

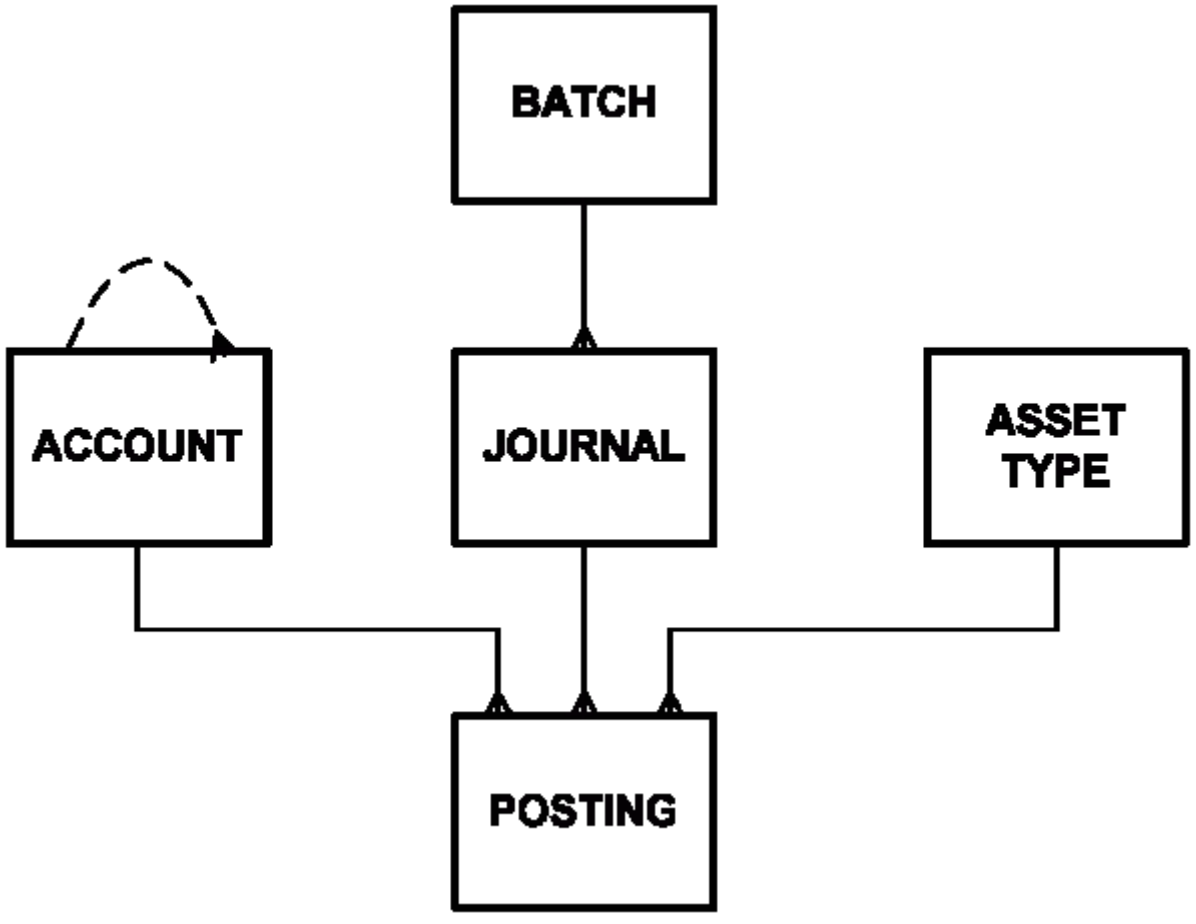
(d) To complete the picture £60 is taken out of the system from the Pattel account with a debt from Pattel and a credit to the Cash book.



At the end of this the Smith account balance is £150, the Pattel account is £40 and the Cash Book is £-190, the negative sum of the other accounts. On this simple basis very complex asset tracking systems can be built.

THE ACCOUNTING DATA MODEL

A simple data model that can be used to represent this information is illustrated below:



The POSTING table contains the actual accounting double entries. Keeping the

figures in one table simplifies the mathematics dramatically. The primary key or part of the primary key of the POSTING table should be a system generated sequence number. It should also be generated in such a way that no gaps can appear in the sequence (through a transaction rolling back, for example). This is part of ensuring that no entries are ever deleted. The BATCH and JOURNAL tables are used to control the input of the double entries to this POSTING table.

A JOURNAL entry is the data representation of any business transaction that will produce double entries — it represents a complete unit of work. I.e. all POSTING entries associated with the JOURNAL entry must be successfully completed or none must be completed. The numerical sum of all POSTING entries associated with a JOURNAL entry must also equal zero (the only way the total value contained in the system can be amended is though the Cash Book). Each transaction described in the example above would be represented by one JOURNAL entry.

A BATCH entry reflects the more physical aspects of accounting data entry. It is used to group together JOURNAL entries into handy 'chunks', for example, a collection of cheques to be entered into the system, a collection of information from an Electronic Data Interface (EDI) or a global business process like the payment of interest to all accounts.

The ACCOUNT table represents the owner of the assets in the system. In most systems the 'pigs ear', recursive, relationship to itself will not be needed. This is only required if a system of 'nominee' type accounts is required where concepts of group ownership are mixed in with individual ownership.

The ASSET TYPE table contains the list of asset types being held in the system. By making the asset type part of the POSTING primary key the system becomes a multi-asset (e.g multi-currency) system. Through use of this, the system can also track the ownership of other 'de-materialised' assets where the actual ownership of any particular item isn't required just the type and quantity owned. Examples of this type of asset includes shares and agricultural or pollution quota. It can also track 'bearer' assets (where, for example, the fact that certificate number 12345 belongs to Mrs X is important).

The above example is shown below as entries in this data model (the column list is kept as simple as possible for clarity).

ACCOUNT				
Id	**Name**			
SMITH	Mr J Smith			
PATTEL	Mr R Pattel			
CASH	The Cash Book			
JOURNAL				
Id *(as used above)*	**Type**			
a	Deposit			
b	Withdrawal			
c	Transfer			
d	Withdrawal			
POSTING				
Id	**Account Id**	**Journal Id**	**Asset Type**	**Amount**
1	SMITH	a	£	300
2	CASH	a	£	-300
3	SMITH	b	£	-50
4	CASH	b	£	50
5	SMITH	c	£	-100
6	PATTEL	c	£	100
7	PATTEL	d	£	-60
8	CASH	d	£	60

The balance of the POSTING Amount column always being zero after every complete JOURNAL transaction (and the software must not allow an incomplete one). From the above the total in the Cash Book adds up to –190 which is the negative sum of the 150 belonging to Smith and the 40 belonging to Pattel.

To expand this system into a multi-currency system a new asset type is added and used. If Mr Smith wished to change £20 into dollars at a rate of £1 to $1.5 the transaction is made though the Cash Book as follows (the required POSTING entries):

Id	**Account Id**	**Journal Id**	**Asset Type**	**Amount**
9	SMITH	e	£	-20
10	CASH	e	£	20
11	CASH	e	$	-30
12	SMITH	e	$	30

ACCOUNTING PERIODS

Our accounting model so far is perfectly sound but could creak under high volumes as we can never delete anything. Most accounting requirements include the concept of an accounting period, maybe monthly, three monthly or yearly, for example. Whilst there is no need to constrain the logical requirement for an

accounting period by tying it too closely to the physical model it provides a useful indication as to where to break flow of the data. The year end is often a convenient point, either the calendar year end or the financial year end. By adding a period indicator column to the POSTING table (and also to its primary key) the data can be split into discreet time chunks that can be validated independently. If in the above example the next set of entries fell into the next accounting period the balances outstanding are brought forward as summary entries in the next period first (leaving out the currency exchange example for simplicity).

Firstly the balances for the previous period are cleared down.

Id	Account Id	Journal Id	Account Peri	Asset Type	Amount
9	SMITH	e	YEAR 1	£	-150
10	CASH	e	YEAR 1	£	150
11	CASH	e	YEAR 1	£	40
12	PATTEL	e	YEAR 1	£	-40

Then the balances are carried forward to the next period.

Id	Account Id	Journal Id	Account Peri	Asset Type	Amount
13	SMITH	e	YEAR 2	£	150
14	CASH	e	YEAR 2	£	-150
15	CASH	e	YEAR 2	£	– 40
16	PATTEL	e	YEAR 2	£	40

After an appropriate period of time the previous YEAR 1 period can be archived and deleted from the system.

AGGREGATING TRANSACTIONS

There are some activities that occur within accounting systems that are large scale changes that affect many or all of the accounts. For example, payment of interest to all accounts as a percentage of their current balance or a global cut of quota by a certain percentage. The end of year example above also falls into this category. The end of year entries should all be done under the same JOURNAL (although it could be argued that separate currencies/assets could be done under separate journals) but it is quite acceptable to aggregate the entries to the Cash Book. The accounting rule requirements will still be met but it will simplify the work and volumes involved by up to 50%. The POSTING entries in the above end of year example then become:

Id	Account Id	Journal Id	Account Peri	Asset Type	Amount
9	SMITH	e	YEAR 1	£	-150
10	CASH	e	YEAR 1	£	190
11	PATTEL	e	YEAR 1	£	-40
12	SMITH	e	YEAR 2	£	150
13	CASH	e	YEAR 2	£	-190
14	PATTEL	e	YEAR 2	£	40

BATCH ENTRY

A batching mechanism is often used to facilitate data entry to this accounting system. In its historic form this is what has happened for the processing of cheques. An account clerk would be given a pile of ten cheques, a batch number and the total value of the cheques. An entry screen would then be used by the clerk to record the cheques as 'unauthorised' entries. At the BATCH level the number of cheques and their total value is recorded and only when the number of items entered and their totals match these figures is the user allowed to commit the batch. Once this activity is complete the batch is passed to another member of staff who checks the contents of the batch (either on-screen or with the aide of a report) and then 'authorises' the batch if it is correct. This process is generally referred to as 'maker/checker' and can be adapted for the manual entry of any primary asset data. The concept of a Batch is also useful for electronic data transfer as it provides a method of aggregation for audit trail and correction processing. As a rule it is sensible to keep the details of 'unauthorised' entries in separate tables from those 'pucker' double entries made by the 'authorisation' process into the POSTING table. If this is done it is also possible to simplify the programming effort by creating tables more appropriate to the business function in hand. For example, in the above cheque entry process the clerk will only have to identify one account. The other account, the Cash Book, is implicit as the movement of money will always be from the Cash Book to the owners account. A CHEQUE table would only require one Account column where as a hypothetical FUND TRANSFER table would need two, a 'From Account' column and a 'To Account' column.

This is where most of the confusion over the concepts of double-entry principles arise. Most people only come across simple paper Cash Books in the normal course of domestic events. In a paper Cash Book used for example, to record a club's funds, only one entry per transaction is required. This is still an implicit

double entry system because, as only one account is represented (the 'club' in this example) the opposite entry can only ever go against the Cash Book — it is only ever a movement into the system (into club funds) or out of the system (paid out of the club funds). This is also where the other main source of confusion occurs. An individual's bank statement will show money paid into the bank account as a 'credit' because the individual is a creditor to the bank — they have that individual's money. If that same individual ran a Cash Book this figure should be represented as a 'debt' — the bank owes the money/is in debt to that individual. The money is out of that individuals system and in the bank's.

CODE DESIGN

The modules required to support a double entry accounting system are best designed using a tiered, Object Oriented (OO), approach. The tiers used are as follows

1. Interface processing (user and electronic)
2. Business processing
3. Database processing

A full design is dependent of the type of system required but the following hypothetical modules could exist in each layer.

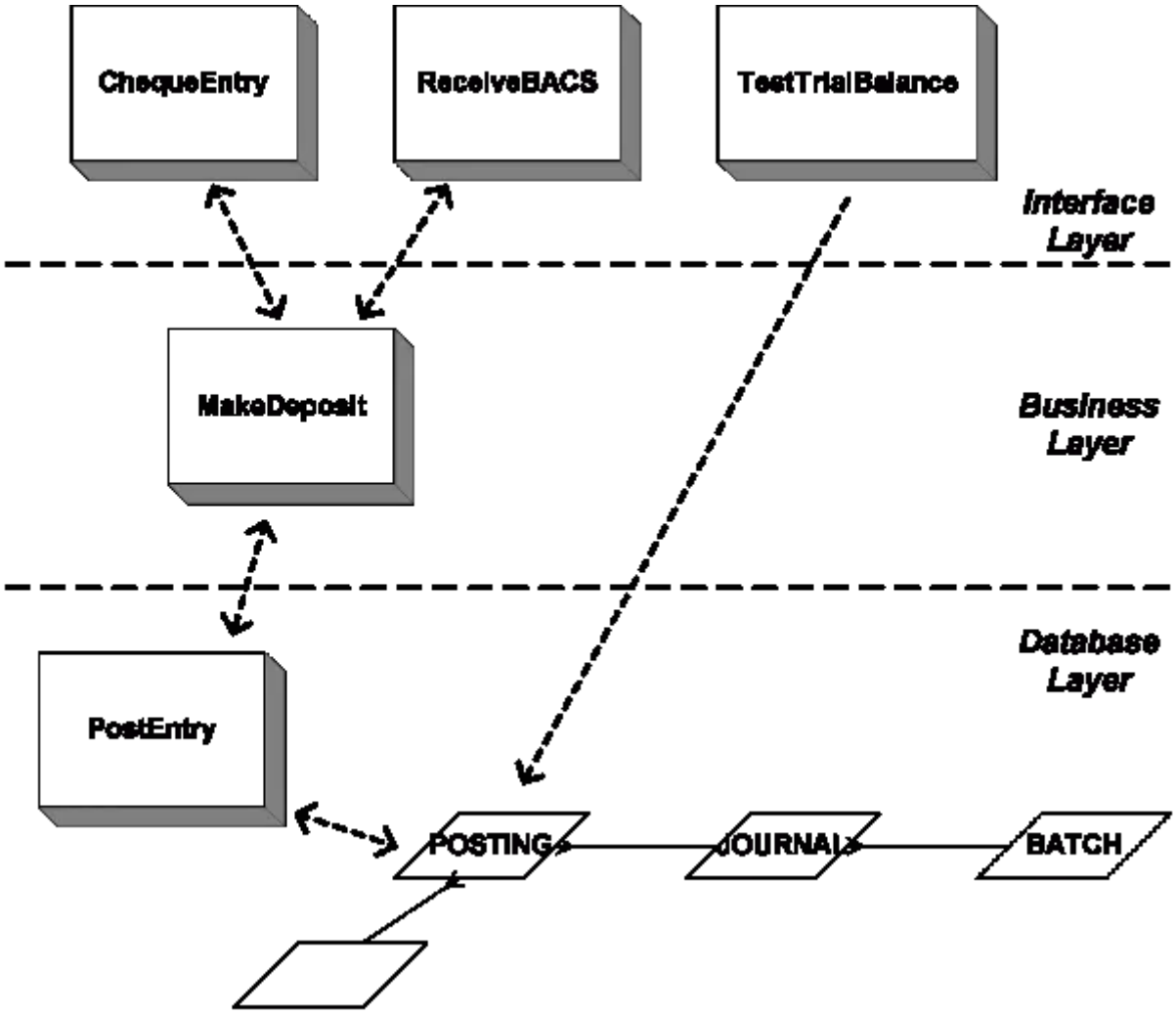
PostEntry: A module to make a basic POSTING entry is required at the database layer. This will insert a record adding timestamps and sequenced identifiers automatically. It cannot delete or update records and no other module will be allowed to update, insert or delete records apart from an archiving module to do deletions. The POSTING table is read-only for all other purposes.

MakeDeposit, MakeWithdrawal, MakeTransfer: Modules to perform basic business process will exist at the Business layer. They will call PostEntry in the appropriate manner.

ChequeEntry and ChequeAuthorisation screens, ReceiveBACS interface module: Modules to link the system to the outside world will reside in the Interface layer. These modules will call the appropriate Business layer objects to perform the business functions. In this way the same processing can be guarantied regardless

of the method of entry because, for example, ChequeEntry and ReceiveBACS would both call the same MakeDeposit module.

This methodology can be used to a lesser or greater degree depending on the complexity of the system and the adherence to 'purist' OO principles. It may be sensible, for example, to allow reports (e.g. TestTrialBalance) to be written to access the tables directly from the Interface layer — rather than building intermediate layer modules to do the work.



THE TRIAL BALANCE

The 'trial balance' is the main method of 'proving' the integrity of any accounting system. If all the entries made into the system have been made in accordance to the double entry rules and are not corrupt in any way then the sum of all these entries will always equal zero. The chance of an equal and opposite, matching, error occurring is so small that it can be completely discounted. The best way to run a trial balance is to start at the highest level and only break the figures down

if a problem occurs. Using the above table structure the parameters for these queries are as follows:

1st Sum the values in the POSTING.Amount column.

Then if that fails

2nd Sum the values in the POSTING.Amount column but broken down by Asset Type and Account Period.

This should indicate where the problem lies. The fact that the sum of every JOURNAL should also equal zero can then be used to track down the POSTING entries that are in error.

JOURNAL ENTITY SUPER-TYPES

The JOURNAL table is the simple representation of an entity or entities that are often more complex and involve super and sub type relationships. The JOURNAL entity could be broken into MATERIALISED and DEMATERIALISED (these might have different attributes because things like certificate numbers or physical location would be required for the MATERIALISED type assets). Alternatively the asset types could define the JOURNAL sub-types resolving them into CURRENCY TRANSFERS, STOCK TRANSFERS, COMMODITY TRANSFERS each having different attributes. Entities with super and sub types can be resolved into tables in one of four ways (these are standard data modelling principles):

1. Have one big table with many optional columns to represent the sub-type attributes.
2. Have a separate table for each sub-type duplicating the columns that are common (those columns that are inherited from the super-type's attributes).
3. Split the entities so that the super-type has one table with its columns and this must be joined to other tables that represent each sub type which contain the sub-type columns only. An 'arc' is placed on the one-to-one relationships between the sub-types and the super-type.
4. Split the entities as in (3) above but duplicate the super-type table columns on [illegible]es.

There are pros and cons for the use of each of these methods. From a double entry accounting perspective it is useful to have the super-type table to provide a natural focus for the POSTING entries — this would rule out option (2). Otherwise option (1) is probably best for a simple accounting system (which is effectively what the example used through out this article is — the JOURNAL Type column acts as the sub-type discriminator column) but option (3) would probably be for a complex system with asset types with wildly differing properties.

Author: Michael Wigley
(Michaelwigley@bcs.org.uk)