Short Notes

Retrospective Update: Data As It Was Believed To Be

Received April 1989, revised August 1991

1. Introduction

This paper describes the approach taken within the Department of Social Security (DSS) by the Local Office Project (LOP), in the design of the computer system for Income Support now operational.

The approach was developed over several months by a combined team of civil servants and design consultants. This paper, however, is the responsibility of the author alone, who is indebted to the Department for permission to publish it, and to the many colleagues, too numerous to mention individually, who contributed to the technical solution adopted.

A cautionary remark needs to be made concerning the word 'adjudication': within the DSS this has a specific legal connotation. In this paper, however, it is used in a restricted technical sense, to denote an irreversible change of state of data, significant to the database design.

A key user requirement of the Income Support system is the ability to make retrospective updates to claimant data, without destroying the original values of the data: a reminder that data is not 'real' in itself, but is simply a representation of what the real world is believed to be at a given point of time. (See[1] for an articulate expression of this point.) In other words, data is nothing more than an assertion about reality, which may only temporarily remain 'true'.

Many authors have drawn attention to the problems of modelling time satisfactorily in a database design. Examples are:[2] the Change Log described by Rosemary Rock-Evans;[3] the time-ordered database modelling facility intrinsic to LEGOL 2.0; and[4] the temporally-oriented data model (TODM) developed by Gad Ariav (New York University). What complicates the DSS requirement is the need to retain in the database multiple versions of the same data 'as it was believed to be' at different points in a time continuum.

TODM regards a database as a sequence of 'slices' taken across a time axis, where each slice is in some respect 'different' to its adjacent slices. What our requirement seems to mandate is not one time axis but two: one representing the 'real' time, and the other the 'reference' time. It is of course rare, in practice, for the same data to be retrospectively updated more than once. But modelling time as two axes rather than one supports the most general case, allowing an item of data to be retrospectively updated more than once, if necessary.

2. Background

In the case of the so-called 'retrospective change-of-circumstance' a claimant notifies the Department retrospectively of illness or of temporary employment that occurred during a period for which a payment of benefit has already been made. Stored data must be 'corrected' and entitlement to benefit re-assessed, but the physical update must be non-destructive in order that the personal circumstances on which the original payment was based remain in the database for the period required to satisfy enquiries and audit requirements.

A second aspect to this is that enquiries on claimant data specify both a 'real' time (the time that the data-assertion refers to), and a 'reference' time (a time at which the data-assertion was believed to be true). A sample query might be 'regardless of what we are now told his savings were on 1st April last year, what did we believe them to be when we last assessed his entitlement on 30 September?'. In this example, the 'real' time is 1/4/88, and the 'reference' time is 30/9/88.

The entitlement of a claimant to an award of Income Support is based on a wide range of information about family conditions, housing situation, previous employment, savings, and so on, all of which the claimant furnishes on a single multi-part claim form. In the supporting logical (entity-relationship) data model, a range of corresponding 'evidence' entity-types is defined, each linked in a conditional one-to-many relationship from PERSON (the claimant) and each through a 'linking' entity to one or more award-components of one or more awards. Figure 1 shows an illustrative structure of just four evidence-types: the Full LOP data model contains many more evidence-types, not included here.

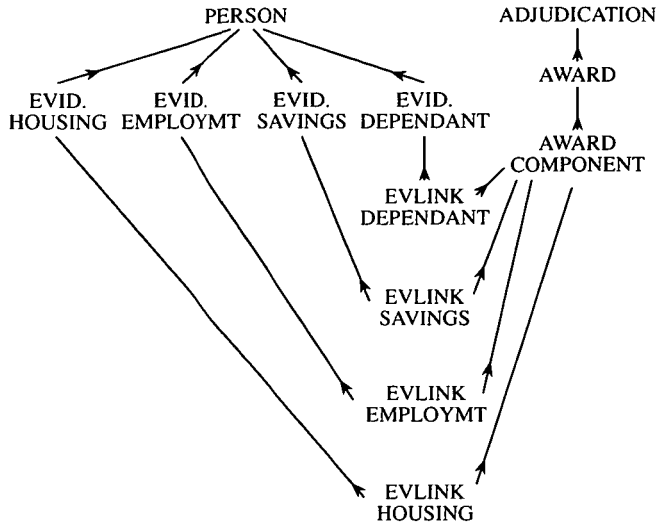

Figure 1. Simplified logical data model.

An important feature of the logical data-model is the many-to-many relationship between EVIDENCE and AWARD: an award is based upon evidences of various types, and a given piece of evidence may be 'used', over time, in a succession of different awards.

Note that the relationships from PERSON to EVIDENCE are of a generic nature, accommodating each of the following:

(*a*) multiple concurrent instances of the same evidence (for example, a claimant with more than one savings account);

(*b*) non-overlapping instances of the same evidence (for example, history of previous housing circumstances);

(*c*) multiple instances of the same evidence, overlapping in time and contradicting each other (as occurs, for example, following a retrospective update).

During the data collection phase (which may extend over more than one database success-unit) destructive update must be permitted, in order to allow the keyboard operator to correct any input errors. But once an entitlement to benefit has been assessed, and an award stored that 'uses' this evidence, any alteration to existing evidence must be non-destructive, and treated as a retrospective change.

The point at which evidence data switches to a non-destructible state corresponds exactly to the event which the current clerical system describes as 'adjudication'. In the terminology of the department, adjudication is performed on a claim, and the 'adjudicated' status is passed on to all the evidence data that is linked to the awards (one or more) resulting from the adjudication.

Since evidence can 'carry forward' from one assessment to the next, and even from one claim to the next, it is frequently necessary that an evidence record be 're-adjudicated'. The reason for a re-assessment need not necessarily be a reported change-of-circumstance: it could arise equally from a change in benefit rules and/or rates. The re-adjudication may involve the termination of evidence previously open-ended, or in some cases the 'amendment' of an evidence, for example by the correction of an amount, a period, or a start- or end-date. It could also involve the insertion or removal of evidence, for example the evidence of a dependent child.

3. Design solution: the standard evidence-header

In order to preserve a simple mapping from 'logical' to 'physical' design, a Cobol record has been defined corresponding to each of the EVIDENCE entity-types.

The design solution employs a standard header-group common to each of the EVIDENCE record-types (see below). At the time

of initial data-capture, the two dates EVID-STRT-DT and EVID-CPTD-DT are always stored, and supplemented at the point of adjudication by EVID-ADJU-DT.

Depending on evidence-type, the remaining data-portions of the evidence records vary in length, from a single field of 6 bytes in one case to some 20 fields totalling 160 bytes in another. A change to one or more of the fields within an adjudicated evidence-record requires a new record of the same type to be stored, with unaltered fields copied forward from the old record to the new. At the same time, the field EVID-END-DT is set on the standard header-group of the old record: this indicates that the evidence ceased to be valid on this date.

Where the change is retrospective, then a different control-field is set, namely EVID-AMDD-DT. In this situation, EVID-END-DT may or may not already be set: either way, it is left unchanged. Whether or not the change is retrospective, the general interpretation is that the data-content of the record is 'believed to be true' up to the date signalled by EVID-AMDD-DT.

It may appear wasteful, at first sight, that unchanged fields from the 'old' record have to be duplicated in the 'new' one. However, this process is not as costly as it might first appear, since volatility is low, and once adjudicated, any given evidence-record is not likely to require amendment more than once in its life.

For ease of processing, the date-fields in the header-group are supplemented by a status field EVID-STAT-TP, carrying the range of values indicated below. Non-mandatory date-fields are ascribed the default value 999999: this default value would be used, for example, as the end-date of any 'ongoing' evidence, such as data about a person's savings.

The items of the header-group are as follows:

EVID-NO	(internal use only).
EVID-STRT-DT	date from which this evidence is valid.
EVID-END-DT	last date on which this evidence is valid.
EVID-ADJU-DT	date this evidence first adjudicated.
EVID-RECD-DT	date of notification to the DSS.
EVID-AMDD-DT	date on which this evidence was terminated or amended by the creation of other evidence of the same type.
EVID-STAT-TP	status of this record (see table).
EVID-VFD-FG	(certain evid. types only) verification status.
EVID-CPTD-DT	date of creation of this record.

Range of values for EVID-STAT-TP

0 Evidence has been adjudicated.
1 Previously adjudicated, now pending re-adjudication in an unchanged state.
2 New evidence, not yet adjudicated.
3 Evidence that has been superseded.

The life-cycle is illustrated in Fig. 2, and the sub-sections that follow outline the alterations that may be applied to the items within the evidence-header during the stored life of an evidence record.

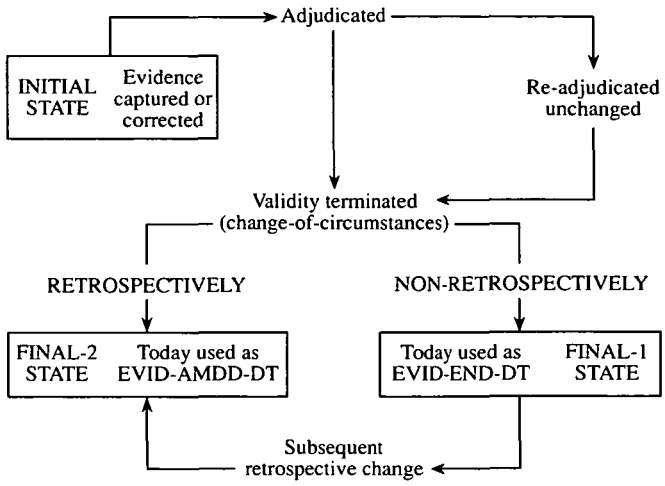

Figure 2. Life-cycle of an evidence record.

3.1. Creation and correction (pre-adjudication)

When data has been collected from a terminal and is to be stored prior to adjudication, EVID-STAT-TP is set to 2. This setting allows any of the date fields in the record to be updated 'in-situ' (i.e. destructively), typically to correct validation failures. The other header items that are mandatory at this state are EVID-STRT-DT and EVID-CPTD-DT. Where an end-date for the validity of the evidence is known, this will be recorded in EVID-END-DT. Otherwise EVID-END-DT carries the default 'open-ended' value of 999999.

3.2. Adjudication

When an evidence-record is first used in an adjudication, the EVID-STAT-TP is set to 0. This setting 'freezes' the data-items of the record to their current values, ensuring that any retrospective update is accomplished only by the creation of one or more replacement records. At the same time, EVID-ADJU-DT is set to today's date. In practice, the data collection and adjudication processes are normally collapsed into a single database success-unit, and the majority of evidence records are already adjudicated by the time they are first committed to the data-base.

3.3 Pending re-adjudication

When an evidence-record, previously adjudicated, is selected for re-use in a later adjudication, but adjudication is to be deferred, EVID-STAT-TP is set to 1. While set to 1, the record is not to be treated as 'adjudicated', but as 'subject to review'. If the adjudication, when it comes, accepts this record, then EVID-STAT-TP reverts to 0. Otherwise (see 3.4 below) EVID-STAT-TP will be switched to 3, indicating that the evidence is 'superseded'. In either case, the value of EVID-ADJU-DT is left unchanged.

3.4 Amendment to termination

A change-of-circumstances implies either: (*a*) that an existing 'open-ended' evidence is to be terminated, or (*b*) that some detail recorded about an existing evidence is to be changed. Additionally, or alternatively, it may be that (*c*) a new evidence record is to be stored. If the change-of-circumstances is retrospective, then the re-adjudication requires that one or more existing evidence records, with an 'adjudicated' status, be marked as 'superseded', by setting EVID-STAT-TP to 3, and EVID-AMDD-DT to today's date (the date of the re-adjudication).

In other words, there are two possible 'final' states for an evidence-record, the normal one where it is simply superseded, and the retrospective one where it is discovered to have been invalid from some earlier date. It is possible for a retrospective change-of-circumstance to affect an evidence already superseded.

4. Design solution: time-related data selection

To answer a query of the kind 'what did we believe the claimant's savings on 1/4/88 to be when we assessed his entitlement on 30/9/88', the logical data structure of Fig. 1 needs to be extended by two 'time-period' entities. The REAL-TIME-PERIOD is a period of time bounded by the EVID-STRT-DT and EVID-END-DT of the attached evidence. The REFERENCE-TIME-PERIOD is the period of time bounded by the EVID-ADJU-DT and EVID-AMDD-DT of the attached evidence. This takes account of the facts:

(1) that prior to EVID-ADJU-DT the evidence was either not known, or not yet adjudicated;

(2) that between EVID-ADJU-DT and EVID-AMDD-DT the evidence was believed to be 'true', and was used, possibly more than once, in an adjudication;

(3) that from EVID-AMDD-DT onwards, the evidence was irreversibly superseded.

Any access to stored evidence-data requires two dates to be supplied – a 'real' date, for matching against REAL-TIME-PERIOD, and a 'reference' date, for matching against REFERENCE-TIME-PERIOD. A value of 3 in the item EVID-STAT-TP is sufficient to reveal that the data was later (i.e. after the reference-time) found to be in some respect 'untrue'.

The importance of the REFERENCE-TIME-PERIOD is that in constructing a picture of the evidence 'as it was believed to be' at the reference-date, it is necessary to exclude any record whose EVID-ADJU-DT is

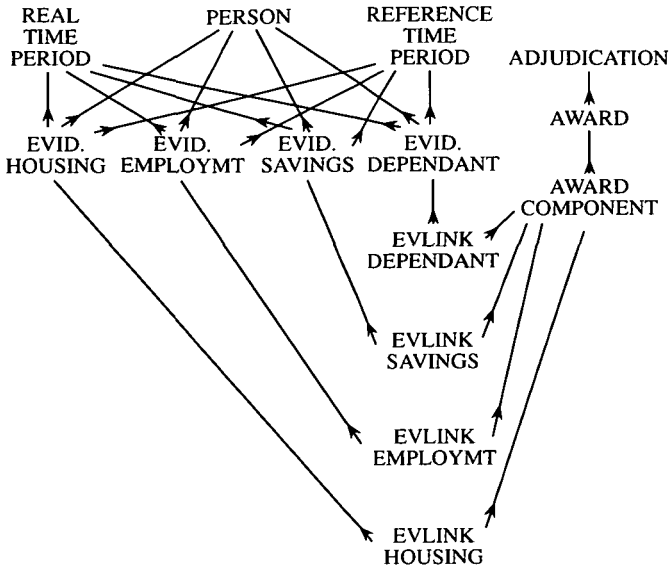

Figure 3. Extended logical data model.

later than the reference-date, or that has an EVID-AMDD-DT that is not 999999, but is later than the reference-date.

5. Conclusions

The design solution outlined in this paper evolved only slowly, after several false starts. Enlightenment occurred with the realisation that the clerical function of 'adjudication' corresponded precisely to 'freezing' all items of a record other than those contained in the standard evidence-header.

Consideration was given to providing REAL-TIME-PERIOD and REFERENCE-TIME-PERIOD as record-types in the physical database structure, to which every evidence record would be physically linked. Our conclusion was, however, that this would lead to unacceptable performance overheads arising from the additional database navigation whenever an evidence record was to be updated. So in the final system, evidence records are chained only to the owning PERSON record, and common routines have been developed that perform the necessary date-matching against candidate records.

The requirement to retain multiple conflicting versions of the 'same' data, as it was believed to be at different points in the past, cannot be unique to the DSS. The design solution described here is now well-proven and is published in the belief that it could be of value to a wider audience.

I. SHEARER

References

1. T. B. Steel, Report on ISO activity in database standardisation. In *Proceedings, International Conference on Databases, Aberdeen, July 1980*, edited S. M. Deen and P. Hammersley. Heyden, London (1980).
2. R. Rock-Evans, *Analysis within the Systems Development Life-Cycle Vol. 1.* Pergamon-Infotech, Maidenhead (1987).
3. S. Jones and P. Mason, Handling the time dimension in a database. In S. M. Deen and P. Hammersley (see 1 above).
4. Gad Ariav, A temporally oriented data model. *ACM Transactions on Database Systems* **11** (4) (1986).

The Reve's Puzzle: An iterative solution produced by transformation

An iterative solution to the Reve's puzzle is produced by largely automatic program transformation from a recursive solution. The result compares favourably with published, manually produced iterative algorithms, both in terms of comprehensibility in its own right, and efficiency.

Received November 1990

1. Introduction

The Reve's Puzzle is an extension of the Towers of Hanoi problem by Dudeney[1] (1907).* The puzzle is posed in terms of 4 stools and a number of cheeses of diminishing size. The rules are identical to those for the Towers of Hanoi.

2. Prologue

Rohl and Gedeon[7] presented a programming solution to the Towers of Hanoi problem with 4 pegs (unaware at the time of Dudeney's book) which used the standard 3 peg solution as a sub-routine as follows:

* Édouard Lucas (1889) mentions a version of Hanoi with 4 or 5 pegs, however, the precise rules are not given. The illustration shows 5 pegs arranged as on a die, with a tower of 16 alternating coloured discs on the central peg, while the text indicates 4 colours for discs *and* pegs. For clarity, it seems best to retain the name *Reve's Puzzle* for the 'standard' 4 peg case, being the earliest unequivocal reference.

```
procedure Hanoi4(n:ndiscs; p1, p2, p3,
p4:pegs);
begin
if n > 0 then
  begin
  Hanoi4(n−F(n), p1, p4, p3, p2);
  Hanoi3(F(n), p1, p2, p3);
  Hanoi4(n−F(n), p4, p2, p3, p1)
  end
end
```

The function $F = trunc((sqrt(1+8*n)-1)/2)$ determined how many discs to move using all 4 pegs, and how many with only 3. The function calculated the ordinal number of the triangular number less than or equal to n.

This function is clearly not invertible, as it performs a many to one mapping, so the parameter n would need to be stacked in an iterative version. Rohl and Gedeon also made the observation that the procedure *Hanoi3* is called with values which decrease by one for each level of the tree of procedure calls of *Hanoi4*, excepting that for a non-triangular number of discs n initially, on one level only the same value is used as the previous one. This level varies on n in sequence; after all the levels have been 'doubled up' in this fashion, the next triangular number is encountered, after which the cycle starts anew. Table 1 shows the values of n for each call to *Hanoi3* from *Hanoi4*.

This is a subtle consequence of the property shown by Roth[9], that the number of moves required for larger towers increases by a number of moves equal to a power of two, a number of times equal to the exponent plus 1. That is, Δ is *1* once, *2* twice, *4* three times, and 2^k '$k+1$' times.

Table 1. The first number indicates the value of *n* in calls to *Hanoi4*, while the list of numbers shows the values of *n* in initial calls to *Hanoi3*.

1	1
2	1, 1
3	2, 1
4	2, 1, 1
5	2, 2, 1
6	3, 2, 1
7	3, 2, 1, 1
8	3, 2, 2, 1
9	3, 3, 2, 1
10	4, 3, 2, 1

We can also see that the number of times for which Δ is $2^{m(n)}$ is repeated is also the same as the depth within the recursive descent on *Hanoi4* that the 'doubling' occurs.

3. The Reve's Puzzle

The above observations lead to the following procedure:

```
procedure (Reve(n:ncheeses);
var position, discontinuity:natural;
  procedure R4(position, skip: natural; s1, s2,
  s3, s4:stools);
  begin
  if position > 0 then
    begin
    R4(position−1, skip, s1, s4, s3, s2);
    if position > skip then
      Hanoi(position−1, s1, s2, s3)
```