PROVIDING TIME-RELATED CONSTRAINTS FOR CONVENTIONAL DATABASE SYSTEMS.

T. Abbod, K. Brown and H. Noble

School of Mathematical Sciences and Computer Studies
Robert Gordon's Institute of Technology
St. Andrew Street, Aberdeen

Abstract

A model for a historical database is described which is based upon time-stamped tuples as the unit of storage. The model includes both physical and logical time-stamps. The technical characteristics of write-once laser discs prevent the use of double logical time-stamps. The model distinguishes version from correction-updates. It is shown that if set-valued attributes are involved the use of null values is unavoidable if back-dated correction-updates are allowed. A method of handling user-defined integrity constraint rules is outlined which involves the maintenance of a time-stamped trace of such rules. Such a trace is necessary for the proper handling of back-dated correction-updates. An outline of a system SIS-BASE is described which implements some of these ideas.

1. Introduction.

Historical databases in which data once recorded are never deleted but are instead time-stamped and augmented by more up-to-date data, have been discussed for some years (1,3,4,5,11,13). The idea has obvious attractions for those applications in which the maintenance of a perfect audit trail, or the rapid retrieval of time-expired data is important. It also holds out the possibility of a more complete model of the real world than is possible with conventional database systems (10).

The storage requirements are so onerous, however, that there have been few, if any, attempts to construct a practical implementation. The advent of write-once laser discs suggests that the idea should be re-examined.

2. Previous work

Various issues have been debated, such as the use of single and double time-stamps (19), the distinction between logical and physical time-stamps (13), the distinction between attributes which are considered fixed in value and those which require progressive amendment over time (19), the distinction between attributes which change continuously with time and those which change according to a step function (12). Theoretical analysts have used a model of a historical database called an "infinite-state" database which records separate (complete) entries for every relation at zero time-intervals (12). As a theoretical construct it forms an interesting model with which other models can be compared. Another model appends time-stamps to individual attribute instances giving rise to a four-way classification of attributes (19). That is, those which have a single fixed value, those which have a single value associated with a double time-stamp (showing the time interval over which the value accurately corresponds to the real world), attributes which have a set of values and those which have a set of time-stamped values.

Additions to the relational algebra have been proposed which take into account the time dimension and the set-value nature of some attributes. The operations proposed allow the user to switch between normalised and de-normalised relations and to bring time-stamps of two attributes into correspondence ("time slice") (6). Snodgrass (16) has complained that there is little agreement on the terminology for most of these ideas and few workers have considered the practical problems of implementation.

3. An outline of our model.

The work we shall describe here is based on a data model which is similar but not identical to that described by Kobayashi (12). Following his terminology we can call it a database with "event" relations. In such a database the existing data is augmented only when a change (an event) occurs. When a single attribute is involved but is part of a tuple, the complete tuple is re-created with the new attribute value, and the whole tuple is time-stamped. Kobayashi has demonstrated the equivalence of such a database with an infinite state database. Later we shall compare this model with a model in which individual attributes are the unit of storage.

We recognise a distinction between updates to the data which are carried out because the conditions in the external or real world have changed and those which are carried out because an error in the existing data has been detected. To avoid confusion we will refer to these as "version-updates" and "correction-updates" respectively. The distinction, which was noted by Klopprogge (11), is, we believe, an important one and it is reflected in our model. We note that while some types of attribute may appear to be fixed in value with respect to version-updates, no attribute is free from a possible correction-update.

The maintenance of a historical database requires the use of two kinds of time-stamp. Following Lum (13) we shall refer to "physical time" and "logical time". Physical time is the time at which an update takes place and its value can be supplied automatically by a system clock. Logical time is the time in the real world at which a change of circumstances occurred which necessitated a change in the database in order to maintain correspondence with the real world. Logical time values must be supplied by the database user but their importance to operations over a historical database system is so great that we have placed their provision and manipulation under system control.

Every data item is associated with two logical time-stamps. These correspond to the earliest and latest limits of the period of time over which the data corresponds to the real world. This is called the "transaction time" by Snodgrass. It is possible to implement a historical database in such a way that each data item holds only its earliest time-stamp limit. To discover the latest limit to its period of "applicability" it is then necessary to consult the next datum in the time sequence. It is also possible to store both limits with each data item. Tansel (19) has discussed the relative merits of each approach. A major disadvantage of the double time-stamp approach, however, is that the value of the second of the two time-stamps (usually) cannot be known at the time a tuple is created. To use double time-stamps therefore, it would be necessary to update the tuple in situ some time after its original creation. We are not aware of any write-once laser disc system in which this is possible and we have therefore adopted the single time-stamp approach. To reduce the inconvenience of having incomplete information stored with each tuple we have introduced "version numbers" which indicate the ordinal position of a tuple in a sequence of such tuples.

To be more precise our model is as follows: Let R be a relation with attributes (A1, A2, A3, ...). Let r be a predicate such that r(a1, a2, a3, ...) is true if the set of data values (a1, a2, a3, ...) is an instance of a tuple within R. We will use suffices to distinguish between tuples in the same relation thus ($a1_1$, $a2_1$, $a3_1$,...), and ($a1_2$, $a2_2$, $a3_2$,..) are both tuples in R. We augment the schema by adding extra attributes: S = a surrogate, T = a logical time-stamp, P = a physical time-stamp, V = a version number, C = a correction number, E = an explanation. Although we shall represent these as part of the tuple, they are not seen by the user and should be regarded as being under system control. We term them "meta-data" because they represent "information about data" rather than simply "data". The values of T and E are supplied by the user.

As a simple example of how version and correction numbers are related to physical and logical time-stamps consider the history of the salary of "joe" shown in figure 1. We can read joe's salary "version-history" (as we currently believe it to be) by reading the top edge or envelop of the chart.

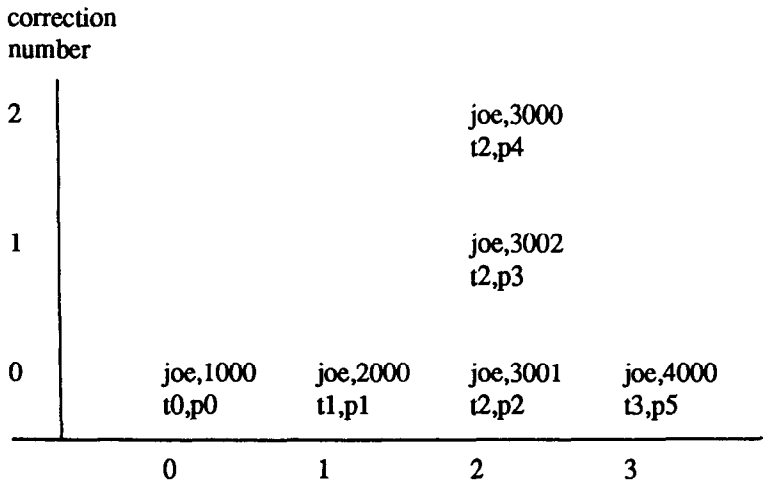

Figure 1

p1, p2, p3, ... physical time-stamps
t1, t2, t3, ... logical time-stamps

We can read the "correction-history" for version 2, say, by reading vertically on that column. In order to read the chart envelop, however, it is necessary for the system to navigate from the top entry in a column to the top entry in the previous column. In a practical database on laser disc pointer fields would be required. A back-dated correction-update, however, requires the creation of a new envelop consisting of simple pointer records inserted to maintain pointer continuity.

Successive Attributes. For an attribute such as "salary" each version succeeds its predecessors and is independent of them. We shall call such an attribute a "successive" attribute. The example above illustrates such an attribute.

Set-Valued Attributes. Consider a relation which holds information of a people's qualifications. If a person acquires a new qualification it is appended to his/her collection. It does not replace previous qualifications. Tansel modelled this by defining a class of attribute which could have a set of values and termed it a "set-valued" attribute. In a conventional database however this state of affairs would be modelled by reduction to first normal form. Tansel suggests extensions to the relational algebra (pack and unpack) which switch the data between these two formats.

Consider the chart of such a relation (figure 2). We dispense with some of the details in this chart but the same conventions hold. Versions are read from left to right and corrections vertically. Here we see that joe has accumulated three degrees (bsc,msc,phd) at different times and that what corresponds to the "current version" is found by reading the chart envelop. Each version is the set of tuples obtained by reading the envelop from the left, up to and including the column bearing the appropriate version number. In effect, each version is embedded in its successors. The model still holds but its interpretation has changed to correspond to the semantics of the type of attribute being amended.

At this point we must consider practical issues of access efficiency. Typically, random access times for an optical disc drive are of the order of 150 milliseconds, or about 10 times the access time for conventional disc drives. The performance of a practical database therefore would be greatly impaired if, in order to retrieve the "current status" it was necessary to make multiple random access retrievals. It may therefore be advisable in some applications that the model should allow storage in the "packed" form - as an unnormalised set of attribute values.

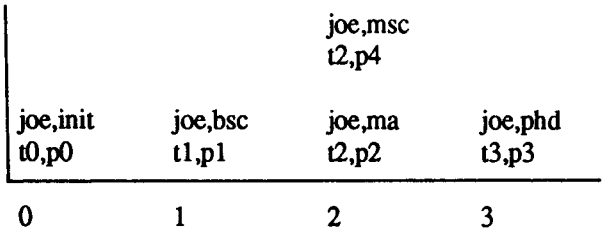

Figure 2

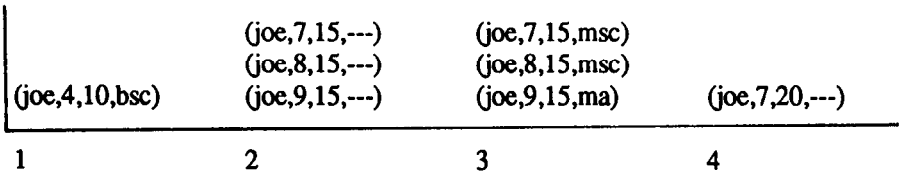

Figure 3

This, however, introduces all of the complications associated with correction-updates which we describe for "cumulative" attributes below.

Cumulative Attributes. Consider an attribute which represents a cumulative total, or some other aggregation of all previous values. We shall call these "cumulative" attributes. A typical example is an attribute which records a bank-account balance. A back-dated correction will propagate forwards in time and require a sequence of correction-updates. In practice an alternative method would be to adopt the time honoured methods of the accountant and make a single correction-update to the most recent version (with an appropriate explanation). This is one of the possible uses of the "explanation" attribute. Such a "solution", however, destroys the integrity of the historical trace and we shall ignore it for the purposes of this discussion.

The final consideration, before we leave this topic, is the treatment of tuples which have more than one attribute of more than one type. Consider the tuple with schema person(name,balance,salary,qualifications). Each one of these attributes could be the subject of an update and our method of identifying a version would be in some disarray if we could not identify which attribute had been updated at any one time.

In figure 3, note the use of null values for the attribute qualification in those columns where it is not the subject of a version update. Without the use of null values (in column 2 for example) we would be gratuitously providing joe with an additional qualification. It is worth noting therefore that null values are obligatory in a historical database which uses the tuple-based model and does not use set-valued attributes. It is still possible to read off the current version for the set-valued attribute qualifications (ie the envelop) provided we ignore the null values.

In the case of a successive attribute (salary) we have successive versions corresponding to columns 1,2 and 4. There was no salary change corresponding to column 3. To avoid wrongly taking column 3 to be a version with respect to salary we require that entries carry some indication of which attribute is the subject of version-update. Furthermore this indicator must be inherited by subsequent correction-update entries. Correction-updates are applied to the appropriate version (or column) and the "knock on" effect (ie propagation of correction to columns on the right) extends only as far as (but not including) the column where the next version update for that attribute occurred. The need for indicators showing which attribute is the subject of update is the main functions of the "explanation" parameter E introduced earlier.

The history of "joe" is still represented by the chart envelop and the model holds for tuples with mixed attribute types.

4. Integrity Constraints

It is not common for a database system to support user-defined integrity constraints but some do (8). The constraints normally consist of rules which are applied whenever an attempt is made to insert a new item of data (or when an existing item is updated). The insertion/update is prevented if the rules are violated. Various strategies for the invocation of integrity constraints have been proposed (2,17) and the relationship between integrity constraints and formal logic has been analysed (9). What does not appear to have been pointed out before, however, is that, if integrity constraints are user-defined they can be expected to change from time to time. In a historical database system the rules should be time-stamped and a back trace of rules should be maintained as for data. It would be inappropriate to apply current integrity constraints to a correction-update being applied to old data, which was originally created when the integrity constraints were quite different. This has considerable implications for the implementation of the propagation aspect of correction-updates. In making a distinction between version and correction-updates Klopprogge (11) suggested that the end-user without special access rights should be able to make only version-updates to the database. Correction-updates would be reserved for "superusers" who could ignore the normal integrity constraints. We suggest that this would not be necessary if correction-updates were subjected to back-dated integrity constraints.

If integrity constraints are user-defined they can not only be expected to change from time to time but they can be expected to be defined incorrectly on occasions. It follows that there should be a set of meta-rules, (which are universal in their application and which are never user-defined and therefore are not subject to time-related changes), which can be used to prevent the insertion of incorrect user-defined integrity constraints (15). Our model supports meta-constraints as well as constraints and uses a classical proof by contradiction method for proving constraints. Our model does not therefore support the idea of correction-updates with respect to integrity rules, only version-updates are allowed.

Meta-constraints are not time-dependent in our model. They are intended to correspond to real world common sense rules such as "an object cannot exist in two locations at the same time". The model does not support user-definition of meta-constraints.

In section 3 we introduced the idea of a predicate $r(a1_i, a2_i, a3_i,....)$ which is true if and only if the tuple $(a1_j, a2_j, a3_j,..)$ is a tuple within the relation R. We now introduce a new relation R' with an associated predicate r' which is analogous to R and r. In concrete terms, R' can be regarded as a "scratch" relation within the deletable file-store which acts as a temporary holding point for tuples submitted for insertion to the relation R which resides in permanent (non-deletable) file-store. There is one R' for every R within the database schema. When we define an integrity constraint we can therefore distinguish between newly submitted tuples and those which are already in the main database and have therefore, presumably, already been passed by the integrity constraint mechanism as valid. In general we write an integrity constraint in the form:

(rule-form-1)

```
over(<time-interval>):
   ( <condition-1>
      => should_be(<condition-2>) )
```

Where: <time-interval> indicates the limits of applicability of the rule. <condition-1> is some condition involving r' , which will have a time-stamp (T). <condition-2> is some condition which can be inferred from condition-1 provided T lies within the time span of <time-interval>.

The introduction of the modal operator "should_be" may be thought an unnecessary embellishment, but it is not only intuitively "comfortable" - most people when asked to describe informally an integrity constraint rule use exactly this construct - but it simplifies the translation of the rule into Prolog (15).

In addition to such rules which are user-defined we add the meta-rules in the form:

(rule-form-2)

```
should_be(<condition-2>)
      at (<time-interval-2>)
& should_be(<condition-3>)
      at (<time-interval-3>)
& concurrent(<time-interval-2>,
            <time-interval-3>)
      => contradiction;
```

It is simple to translate a rule in the "should_be" form into another with the form:

(rule-form-3)

```
over(<time-interval>):
  <condition-1> & not(<condition-2>)
            => invalid;
```

and in this form efficient checking of the integrity of a tuple can be preformed. In rule-form-1, however, the validity of the rules themselves can be checked by the meta-rules by the classical proof by contradiction.

5. SIS-BASE and SIS-BASE-2

The model we have been describing arose out of a project to develop a system called "SIS-BASE". This system embodies many (but not all) aspects of the model we have already described. In addition it has one important feature which distinguishes it from other attempts to construct a temporal database. Instead of trying to develop a completely new database system SIS-BASE is what might be described as a temporal front-end to INGRES. Stonebraker (18) has described some temporal extensions to INGRES but in our case the additional features are separate and distinct from the existing features of INGRES. This approach was discussed some time ago by Ariav (1). Another prominent feature of SIS-BASE is that Prolog is the implementation language of the front-end. This makes possible the convenient specification of integrity constraints. Our hope is that it will eventually become an easily ported front end to a variety of database systems.

One drawback is that the association with INGRES curtailed our freedom with respect to the model and predisposed us to a tuple-based model. To free ourselves from this restriction and to address the problems of a practical system making use of write-once laser discs we have begun to develop a model for a second system (SIS-BASE-2).

6. SIS-BASE: An outline of the system

SIS-BASE holds the "historical" information in tabular form. We shall refer to this table as the "meta-information table" (MIT). There is one table for each conventional relation in the database. Each table has the structure: (key, physical-time, logical-time, status pointer, explanation). "Key" is the value of the primary key for a given tuple within the relation. "Physical-time" is obtained from the system clock. "Logical-time" is provided by the user but the SIS-BASE prompts the user for its value. To determine which attribute has changed between versions it is necessary to examine the "explanation". The "status" indicates whether the tuple concerned is "initial", "historical" or "terminal". The "pointer" (a surrogate index number) identifies the previous version in the prolog database. An "initial" tuple is one which is the starting point of a sequence of versions. A "historical" tuple has both antecedent and possible descendent versions and a "terminal" tuple brings a series of versions to a close.

An interesting point which arises in a temporal database concerns the re-use of a key value after it has been terminated. An example would be the re-use of an employee works number some years after the original employee with that number had died. SIS-BASE has a special indicator (under status) which means "was terminated - is now re-initialised"). Allied to this is the possible update of a key. It is sometimes necessary to re-structure a database and re-define keys. The need for this occurs for example when companies merge and customer numbers are re-assigned. In a conventional system the database is simply re-structured and no anomalies arise, but in a temporal database with its need to maintain continuity with past history, the old keys values must be retained and linked in some way with the new values. The surrogate introduced earlier is designed to

maintain this continuity. It is hidden from the user. By definition surrogates do not undergo update.

SIS-BASE accepts definitions of relation schema (attribute names, attribute types and tuple structures) and stores these. It also converts the schema with meta-attributes appended into the appropriate format and submits them to INGRES. SIS-BASE is therefore able to prevent erroneous attempts to insert data into non-existent relations without invoking INGRES facilities.

The user specifies the target relation and enters the data. Integrity constraints are invoked and these may be of two types - referential constraints which check for the existence of cross-reference keys etc in other relations (7,14) and assertion constraints which require the specification of rules. These rules are coded in Prolog and can therefore go well beyond the normal facilities of INGRES. Referential constraints are not available in INGRES. If the tuple passes these checks it is translated into the appropriate code in QUEL and submitted to INGRES.

The deletion of a tuple is converted into the insertion of a tuple with status = "termination" which becomes the current version. The amendment of a tuple is converted into the insertion of a new current version of the tuple. It inherits the un-amended data from the previous current version. SIS-BASE does not distinguish between version and correction-updates. Version-update also becomes an insertion process and integrity constraints are invoked.

7 The Specification of Integrity Constraints.

To illustrate the conversion of a constraint rule in rule-form-1 into Prolog consider the rule that no salary should exceed or equal 25000 during the time interval tt1 to tt2. We assume that our database contains a relation R with schema (N,S) or (Name,Salary) and that we have extended it by the addition of a logical time-stamp attribute T.

```
over(tt1,tt2):
   r'(N,S,T) => should_be(lt(S,25000))
```

Where lt(X,Y) means "less_than(X,Y)".

We introduce a new predicate with form "should_be_lt(X,Y,T1,T2)" which is true if lt(X,Y) during the time interval T1-T2. That is, we combine the modal operator with the predicate lt to produce a composite predicate. The use of the composite predicate makes it much easier in Prolog to write rules for (say) the transitivity of the relation less-than without generating infinite recursion. We can now write:

```
should_be_lt(S,25000,tt1,tt2):-
          r'(N,S,T),within(tt1,T,tt2).
```

where within(T1,T,T2) is true if T lies in the time interval T1- T2.

This in turn translates into a rule in rule-form-3

```
invalid :- r'(N,S,T),
      within(tt1,T,tt2), not_lt(S,25000).
```

where not_lt ("not less than") is the converse of should_be_lt.

This representation of the rule conforms to the structure of the rules described by Weigand (20) which in turn was shown by him to conform to the ISO-report version of Interpretative Predicate Logic.

r' is the predicate associated with the scratch relation R' which is held in SIS-BASE but is not "seen" by INGRES. The predicate r however which is necessary for the definition of some constraint rules invokes in SIS-BASE the retrieval of the appropriate relation from INGRES to make it available to the Prolog inference engine.

In a historical database the concept of "next" is important. We might for example wish to specify that after a person has been employed in department d1 he or she must be employed in department d2 "next". When tuples carry only single time-stamps it could be difficult to define the concept "next". One method of showing that a tuple r(_,T2) is the "next" tuple after r(_,T1) is to search and fail to find another tuple r(_,T) such that within(T1,T,T2) is true. This is a very clumsy mechanism which is eliminated by the use of

version numbers. r(_,T2,V2) is the next tuple after r(_,T1,V1) if V1 and V2 are version numbers and V2=V1+1. The use of double time-stamps also simplifies the definition of "next" but violates our principle that updates to a tuple are disallowed once it has been inserted into the database.

8 Discussion

The initial object in mind was the provision of a portable temporal front-end to a conventional database system and to make available the full power of Prolog as a means of specifying integrity constraints (including temporal constraints) (SIS-BASE). The development of a temporal model was a necessary adjunct to this work and in the event, the attempt to implement SIS-BASE resulted in a considerable clarification and refinement of the model. Some of the results were a surprise to ourselves. We did not expect that the need for null-values would arise from the tuple-based event model we have described but we believe that the result is important. In a practical database the designer would no doubt normalise the data so that set-valued attributes formed a relation of their own thus minimising the need for null values, but the need cannot be eliminated in a relational database if one concedes that every relation needs a primary key and that no attribute is ultimately free from the need for updates. A model which relied on pointer connections rather than foreign keys to link tuples would avoid this problem.

A second aspect of the work to which we draw attention is the use of version numbers to replace the double time-stamps favoured by some workers, a decision forced upon us by the assumptions we made about the characteristics of the non-deletable file-store. Version numbers simplify the specification of integrity constraints in Prolog (as do double time-stamps) without introducing a degree of update dependency. Double time-stamps based on logical time values which are supplied by the user are possible targets for version-update and this would then require two tuples to be updated in this manner to avoid possible gaps or overlaps in the time continuum.

We have proposed the use of pointers to allow navigation between the various versions and their respective corrections (and not to replace the use of keys). These are unnecessary in an environment like SIS-BASE which makes heavy use of Prolog but would be necessary to minimise access times in a system based on large scale file-stores.

What emerges from our analysis is the complexity of the procedure necessary for a correction-update which is applied to time-expired versions, and the significant overheads in maintaining these pointers. There is the additional complexity introduced by the fact that the requirements for such updates vary with the type of attribute being updated. Cumulative attributes require a procedure which propagates throughout the database, set-valued attributes do not and successive attributes require a "knock-on" procedure which terminates at the next version which modified that particular attribute.

Some comment on the relative merits of the tuple-based model described here and an attribute-based model would be appropriate. Both models are subject to correction-update propagation problems. It is obvious that the tuple-based model involves the redundant storage of data which an attribute-based model avoids. The attribute-based model also avoids the need for null values which we identified in section 3 but increases the number of pointer fields. The most important factor is the typical proportion of a tuple (in terms of storage volume) which is updated. If attributes are updated singly and independently of each other and no attribute is large in proportion to the whole tuple then the attribute-based model is clearly the best choice. In other circumstances it may be advantageous to trade storage space for access speed. A practical system can be based upon a hybrid model in which the unit of storage would change dynamically.

Perhaps the most important aspect of our analysis is the need for time-stamped integrity constraints. These are required to control back-dated correction-updates but could also be used to reduce the search space of some queries. Examination of the historical trace of integrity constraints could be used to eliminate certain periods of time from the search because the condition sought was impermissible during those periods.

Acknowledgements

T. Abbod's research is funded by the Scottish Education Department. We would also like to thank the Governers of RGIT for providing the facilities with which this work was carried out.

References

1. Ariav G. , Clifford J. and Jarke M. "Time and Databases", ACM-SIGMOD, SIGMOD Record, Vol. 13, No. 4, SAN JOSE, 1983.

2. Brodie M. L., "The Application of Data Types to Database Semantic Integrity", Information Systems, Vol. 5, 1980.

3. Bubenko, J. A. "The temporal dimension in information modelling", Architecture and Models in Database Management System, (G. M. Nijssen ed), North Holland, 1977.

4. Castilho J. M. , Casanova M. A. and Furtado, "A Temporal Framework for Database Specifications", Proc. of the 8th Int. Conf. on VLDB, Mexico City, 1982.

5. Clifford J. and Warren D. S., "Formal Semantic for Time in Database", ACM-TOD, Vol. 8, No. 2, 1983.

6. Clifford J. and Tansel A., "On An Algebra for Historical Relational Databases: Two Views", Proc. of ACM-SIGMOD, Austin, Texas, 1985.

7. Date C. J., "Referential Integrity", IBM General Products Division, 555 Bailey Avenue, San Jose, 1981.

8. Frost R. A. and Whittaker S., "A Step Towards the Automatic Maintenance of the Semantic Integrity of Databases", The Computer Journal, Vol. 26, No. 2, 1983.

9. Frost R. A. "Formalising the Notion of Semantic Integrity in Database and Knowledge Base Systems Work", Proc. of the 5th BNCOD, Kent, 1986.

10. Jones S. and Mason P. J., "Handling the Time Dimension in a Database", BNCOD-1, Aberdeen, 1980.

11. Klopprogge M. R. and Lockemann P.C. "Modelling Information Preserving Databases; Consequences of the Concept of Time", Proc. of Int. Conf. on VLDB, Florance, Italy, 1983.

12. Kobayashi I., "Validating Database Updates", Information Systems, Vol. 9, No. 1, 1984.

13. Lum V., Dadam P., Erbe R., Guenauer J. and Pistor P. "Designing DBMS Support for the Temporal Dimension", ACM-SIGMOD Record, Boston, 1984.

14. Noble H., "Occurrence Dependencies in a Relational Database", Proc. of 3rd BNCOD, Leeds, 1984, Camb. Univ. Press.

15. Noble H. and Abbod T. "Meta-rules and Semantic Integrity Constraints in Databases" Proc. of 5th BNCOD, Kent, 1986, Camb. Univ. Press.

16. Snodgrass R. and Ahn I. "A Taxonomy of Time in Database", Proc. of ACM-SIGMOD, Austin, Texas, 1985.

17. Stonebraker M., "Implementation of Integrity Constraints and Views By Query Modification", ACM-SIGMOD 1875.

18. Stonebraker M., "Adding Semantic Knowledge to a Relational Database System", in Conceptual Modelling, (ed: Brodie, Mylopoulos and Schmidt), Springer-Verlag, 1984.

19. Tansel A. U., "Adding the Time Dimension to Relation Model and Extending Relational Algebra", Inf. Systems, Vol. 11, No. 4, 1986.

20. Weigand H. "Conceptual Models in Prolog" in Database Semantics (ed: Steele and Meersman) pp 56-69, Elsevier Sc. Pub. IFIP 1986.