DECEMBER 1988 VOL. 11 NO. 4

a quarterly bulletin of the
IEEE Computer Society
technical committee on

Data Engineering

CONTENTS

SPECIAL ISSUE ON TEMPORAL DATABASES

THE INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS, INC.
IEEE

IEEE Computer Society

The LOTUS Corporation has made a generous donation to partially offset the cost of printing and distributing four issues of the Data Engineering bulletin.

Letter from the Editor–in–Chief

This issue, guest–edited by Rick Snodgrass, closes out 1988 for us. Larry Kerschberg has come aboard as the new chair of our TC, and has appointed Stefano Ceri as vice chair. Larry is also pushing the idea of membership dues for our TC, as a means of funding our activities and of solving the problem of erratic distribution of our bulletin. He will report on the progress of this effort after the IEEE TAB meeting.

Our 1989 publication plans are as follows. Sushil Jajodia and I will edit an issue on Databases for Parallel and Distributed Systems for March. The issue will consist largely of selected papers from the International Symposium on Databases for Parallel and Distributed Systems to be held in Austin on December 5–7, this year. The June issue will be guest edited by Ami Motro of USC; it will be on fuzzy queries and incomplete databases. Michael Carey will put together the September issue, in cooperation with the organizers of the post–SIGMOD–89 Workshop on Persistent Programming Languages to be held in Oregon. The issue will consist of selected papers from the workshop. Roger King will close out 1989 with an issue on graphical interfaces to database systems for December.

During the past several years, we have experienced problems with the distribution of our bulletin to members of our TC: Some received nothing; while some received only some of the issues. David Barber of the Computer Society has agreed to handle all enquires concerning distribution of the bulletin: please direct any complaints about distribution or requests for back issues to him. In my interactions with him, I have found him very reliable.

Won Kim
Austin, Texas
November, 1988

Letter from the Editor

This special issue concerns databases that capture the evolution over time of the enterprise being modelled. In a very real sense, the first databases, those hieroglyphics describing the pharoah's inventory of grain scrawled with great effort on pyramid walls, already integrated time. In current terminology, these were tuple-timestamped, rollback databases (a common phrase describing the transaction time of rollback databases, "the past information is unalterable, as if written in stone," certainly applies here). The first historical database, Homer's *Odyssey*, was infinitely more malleable, changing in subtle ways with each retelling. While the three-dimensional view of an historical relation was introduced somewhat more recently (by Fred Brooks in his Harvard doctoral dissertation, 1956), temporal databases were not studied until the early 1970's, when Wiederhold and others included time support in their medical information systems. By the end of that decade, both the research and business communities were becoming comfortable with database theory and practice, respectively (!), and a plethora of extensions soon emerged: engineering DB's (CAD/CAM/CASE/CAE), geographic DB's, image DB's, knowledge DB's, object-oriented DB's, radiology and hospital DB's, spatial DB's, statistical DB's, and, as popularized in the scientific press by the significant Clifford and Warren 1983 TODS paper, historical DB's (to be fair, other, earlier articles set the stage for the ensuing surge of interest in time in databases). Mindful of the precedent set by the definition, formalization, *then* implementation of the relational model in the early 70's, and cognizant of the daunting philosophical, linguistic, and even physical difficulties with defining time, researchers focussed on formalizing, as carefully as possible, extensions or replacements for the relational model, algebra, and calculus.

Healthy controversy developed almost immediately. Should the model incorporate objects? The standard relational model does not, but an argument can be made that object identity is crucial when time is modelled. Should the algebra be entirely "syntactic", or should it require various integrity and normal form constraints, and hence be considered "semantic"? What aspect(s) of time should be modelled? Ilsoo Ahn and I have argued that both valid and transaction time should be supported by any database calling itself temporal. Should first normal form be considered a prerequisite, as in the standard model, or are non-first-normal-form proposes superior? Both approaches are evident in the papers in this special issue. Should tuples or attributes be time-stamped? These difficult, fundamental questions are debated in the literature, in the hallways at conferences, and even in referee reports. Attempts to address these questions have resulted in, at last count, 11 algebras, 8 calculus-based query languages, and many data models.

Over the last several years, the focus has started shifting towards implementation, with several prototype DBMS's in place, and movement, but not yet products, by several computer companies. This shift signals a maturing of the discipline; the bibliography in this issue provides another data point.

The seven articles gathered in this special issue represent the current thinking of some of the most active participants in the area. Many of these papers reflect ideas in evolution. Consult the conference proceedings in 1989 and the journals in 1990 (and 1991 and ...) to see the approaches and concepts these papers engender. I expect that the debate will remain lively for quite some time, with controversy yielding to consensus, and then replaced by nascent controversy concerning different issues. All that we know about conventional databases, which is substantial, must be reexamined when time is introduced. The learning curve is steeper this second go-around, but there is still much to do before we catch up.

I would like to thank each of the authors for their contribution, and for meeting tight time and space constraints.

Richard Snodgrass
October, 1988

HISTORIES and VERSIONS for MULTIMEDIA COMPLEX OBJECTS

Michel E. ADIBA

IMAG- LGI -Grenoble University
BP 53X - 38041 Grenoble Cedex- France
Tel. (33) 76 51 46 27
E-mail (UUCP) adiba@imag.imag.fr

Abstract: After reviewing previous work on time and histories in multimedia DBMS's, we define a relevant set of concepts in order to capture and to model semantics associated with the evolution of multimedia complex objects. By interpreting the term "object" in its very broadest sense (i.e. identifier, type, schema, value, methods, etc.) and according to different modelling approaches (e.g. Object Oriented), we characterize histories and versions.

INTRODUCTION:

A lot of work has been devoted for integrating time notions into a DBMS and with facilities to describe and manipulate histories and versions of objects [Anderson 81, Anderson 82, Boulour 82, Clifford 83, Dadam 84, Klopproge 81, McKenzie 86, Snodgrass 85, Snodgrass 86]. The notion of Multimedia Complex Object (or MCO, in short) is now emerging in several new applications (Office Automation, CAD, Computer-Aided Software Engineering (CASE), Geographic, Medical, etc.). In this framework, it is also important to capture and to keep track of object evolution over time. Up to now this problem has not received a complete and satisfactory solution mainly because several distinct models have been proposed for MCO's but also because it is not clear what are the main concepts which concern object evolution [Adiba 85, Katz 83, Katz 86, Kemper 87, Shoshani 86]

So far several approaches have been proposed to model MCO's. First, complex object models have been derived from Non First Normal Form Relations (NF2), which are extensions of the relational model. Second the semantic models have been proposed that associate the description of objects together with their semantic aspects (e.g. extensions to the Entity-Relationship model). Third, Object Oriented models have been proposed.

Our work in this area began around 1983 when we developed models and prototypes in order to deal with MCO's in Office automation and CAD applications. We made several propositions in order to incorporate in a generalized DBMS facilities for managing time and histories. These propositions are summarized in Sections I and II. Our current research is now considering an object oriented approach to the problem of MCO's evolution. By "object", we have to consider not only value but also identifier, type (or schema) and methods (procedures). Also, histories have in general been considered linear, and for several applications (e.g. CAD, CASE), we need to consider branching histories to support versions and alternatives as well. In section III we characterize basic notions in order to integrate in the future, object-oriented DBMS's, specific mechanisms to handle versions of objects.

I - TIME AND HISTORIES IN DATABASES:

In the framework of the TIGER project on Multimedia databases we made several propositions in order to introduce time and historical data in new DBMS's [Adiba 85, Adiba 86, Adiba 87a]. The starting point was to define a type time which consists of a six-tuple sequence: (year, month, day, hour, minute, second). This corresponds to (discrete) instants of time according to the Gregorian calendar. A special key-word "now" refers to the time now, for instance (1988, 08, 27, 13, 20, 32). Several other types related to time concern: (1)*duration* (i.e. a given number for each of the six elements mentioned before). For instance

four days and twelve hours can be represented by (0, 0, 4, 12, 0, 0). (2) *time interval*, for instance [t1,t2] where t1 and t2 are respectively the beginning and the end of the interval. (3) *periodic elements* to modelize periodical activities. All these extensions are available through specific commands of the data definition and manipulation language called LAMBDA [Velez 85]. The time types considered here allows the user to model and to handle valid time [Snodgrass 85].

In order to introduce different kinds of histories in the TIGER data model we consider a database object V which takes, over time, successive values. Up to now, we are not defining more precisely what we mean by object. Let us denote by **(vi,ti)** the pair **(value,time)** for V: at time **ti** V has value **vi**. Hence, a sequence [(v1,t1), ...(vi,ti)] for ti <now is the history of V and (vc, now) denotes the current value. We proposed three notions in order to characterize different kinds of histories:

1) **Periodicity of the ti,** i.e. the sequence of times we want to consider for the vi values. Note that this periodicity may not be related to a change of value.
2) **Modification**: when value v is changed to value v', this modification may or may not give, in a given history, a new version for V. It is the database user's responsibility to define precisely when a change must be incorporated into the history.
3) **Persistency** of values in the database. Theoretically we can consider that successive versions are stored in an "infinite" space. However, the user may want sometimes to keep only the last **n** values. In this case n is the persistency.

A first approach for building histories is to let the user decide when he/she wants to keep a new version after one or several modifications took place on a given object. In this case, we speak about **Manual Version History (MVH).** A special DML statement: "GENERATE VERSION" can be applied to such an object in order to keep several versions.

Second, if we want an automatic treatment for histories, we use an "**each**" clause in the object schema to define the periodicity and in this case, we speak about a **Periodical Version History (PVH).** In a **PVH**, if an object is modified within the period, this modification affects the current version. New versions are only generated at the end of each period by putting in the history a copy of the current version. Third, when there is no manual nor each clauses, this means that we want to store successive versions and then we speak about **Successive Version History (SVH).**

The database administrator is provided with special DDL commands in order to declare, in the schema, the appropriate kind of history for a particular object type. This can be done at the attribute level or at the entity or relationship level [Adiba 86]. In this case, we said that a dynamic object type is defined by opposition to static objects where only the latest value is kept in the database. The DBMS handles histories according to their specific semantics but the **ti's** which are recorded corresponds to physical time, i.e. the time where the modification took place on the object.

The DML provides also special statements for history management i.e. queries. [Bui 86] We consider two different classes of queries on histories corresponding to absolute or relative times. The key-word **version** is used in order to refer to specific versions:
*<u>*Absolute time :*</u> (i) Version of V at time <t> ? ; (ii) Version of V in the interval [t1,t2] ?; (iii) Version of V after (before) time <t> ?.
* <u>*Relative time :*</u> (i) Last (current) version of V? ; (ii) N last (first) versions of V ?; (iii) All versions of V?.

These functionalities lead to specific statements in the query language [Adiba 86] or to temporal query languages [Gadia 85, Snodgrass 87, Navathe 87]. In general, histories are not updatable but sometimes it is necessary to make corrections in order to record consistent data.

For that we provided a special operation.

II- SNAPSHOTS, ALBUMS and MOVIES

The concept of snapshot allows to capture some database states by storing the result of a query on some source objects, as a new database object [Adiba 80, Adiba 81]. This object is read-only and the refresh operation permits to reflect on the snapshot changes made on the source objects. In the framework of generalized databases, we consider snapshots as a new object type, at the same level as entities and relationships. We define the **snapshot** operator which allows to derive information from existing database objects. More generally, let us denote by QUERY(T1,T2,...,Tn) a query where the Ti, $1 \leq i \leq n$ denotes database objects. The snapshot definition may be formally defined as:

type <snapshot-name> : snapshot (<attribute-list>) AS QUERY(T1,T2,...,Tn)

We can create, delete, query a snapshot, but also refresh it: this is the only operation which can modify a snapshot. We consider two possibilities:

1/ REFRESH <snapshot-name>

2/ REFRESH <snapshot-name> AT <time-value>

The first one is an immediate refresh i.e. the DBMS replaces the snapshot content by the evaluation of the snapshot definition query. The second command depends on the value of <time-value>, say **t :**

* **t** is in the past, and we want to see the snapshot "as of" t. For doing this, the DBMS must insure that all the Ti involved in the query are historical and correspond to information known at time **t.** If it is the case, then the snapshot can effectively be built and stored. Otherwise, the DBMS rejects the operation.

* **t** is in the future and we have a deferred refresh. At time t the DBMS will evaluate the definition query and replace the snapshot content.

Let us now consider dynamic aspect of snapshots: as for any database object, it might be interesting to consider successive versions of a given snapshot, i.e. to make snapshot histories. The refresh operation is the only one which can affect the snapshot content. Refresh allows to reflect the database state at a given time. Between two refresh operations, the snapshot content does not change despite of changes made on the other database objects from which the snapshot was built. Many applications require periodic refresh of snapshots: each day or each month (e.g. take the values of banking accounts each month).

As a consequence, we apply our history concept to snapshots, defining different kinds of **dynamic snapshots (or DS).** These different kinds will differ by the periodicity of refresh time, the type of version (Manual, Periodical or Successive), by the persistency of values and finally by the associated time format. More generally, the syntax for dynamic snapshot definition is the following : type <snapshot-name> : dynamic snapshot (<attribute-list>) <historical-structure> AS QUERY(T1,T2,..,Tn)

Here, types T1,T2,..,Tn may be static or historical. In other words a DS may be defined on the base of other histories. The historical structure was explained in section I and in [Adiba 86]. Because we defined three different kinds of histories we have here to consider :

- Manual Version Dynamic Snapshot (MVDS)
- Periodical Version Dynamic Snapshot (PVDS)
- Successive Version Dynamic Snapshot (SVDS)

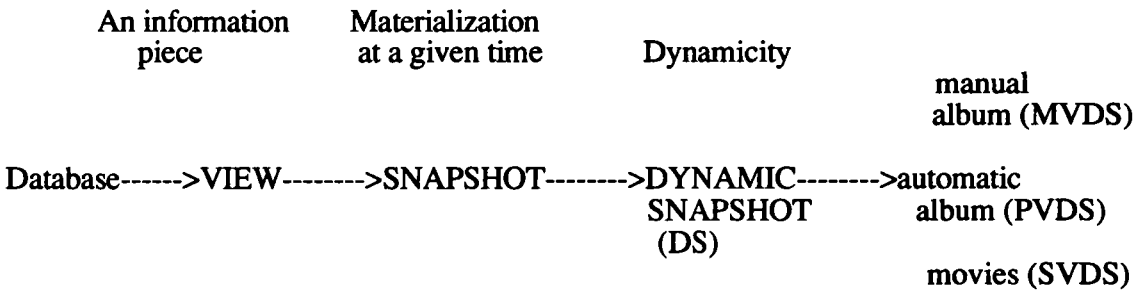
For a MVDS, a new version is generated by the GENERATE-VERSION command which makes a copy of the snapshot value and stores it into the history area. The REFRESH command can modify the snapshot contents but does not generate a new version.

For a PVDS, the periodicity of refresh is indicated by the **'each'** clause (a year, a

month,...). If there is no '**manual**' nor '**each**' clause, we have a SVDS. For this kind of history, refresh has to be made after each modification on T1,T2,..,Tn.

The '**last**' clause indicates persistency of the DS. For histories which are not snapshots a persistency of zero corresponds to a static type and all modifications are made on the current version without keeping versions in the history area. However, the snapshot semantic may take into account the persistency of zero for SVDS or PVDS. For instance, we may want the system to refresh a snapshot each month (the refresh operation is made on the current version), in this case the snapshot is dynamic but has a persistency of zero. We use the key-word '**last only**' for defining a dynamic snapshot with persistency of zero.

Roughly speaking, a PVDS is a collection of snapshots taken at different moments. It gives an automatic album of database portions. In the same way, a MVDS is a manual album. A SVDS reflects all the database changes on the source types T1,T2,..,Tn and so it corresponds to one movie on the database. The following figure illustrates the evolution of concepts : view, snapshot, album and movies.



The evolution of concepts : view, snapshot, album and movies

The history management mechanism suggested in [Adiba 86] remains the same for dynamic snapshot. More precisely, storage spaces for current and old versions are separated. For each dynamic snapshot refresh, instead of destroying the old version, we make a copy and store it into the history area. Queries on dynamic snapshots allow to see some database portions as they were in the past.

Refreshing movies (i.e. SVDS) can be a costly operation. A straightforward, but brute force method consists in replacing the old content by the re-evaluation of the definition query. However, if few objects have been changed in the source elements, this method is not adequate. In other words, an efficient refresh mechanism for a snapshot S should detect modifications on source types which do not affect S. Several methods have been proposed for this problem [Adiba 87a, Koenig 81, Lindsay 86].

III- HISTORIES and VERSIONS OF OBJECTS

The notion of object is related to the data model we consider. In the n-ary relational model, the history notion can only be applied to the relation level. This means that one must be able to (re)build all instances ri that a given relation R had accross time [Lum 84, Dadam 84]. In the E-R approach, we have shown that the history notion can be seen differently. Each entity instance has an internal surrogate and eventually several attributes. By defining some attributes as dynamic types it is possible to maintain a history of all the successive values taken by these attributes, for a given entity instance. However, we did not allow histories of histories [Adiba 86].

For multimedia, tree-structured documents it is not always necessary to store all successive versions of a complete document but only of some of its subparts. Our approach

allows us to define dynamic portions of multimedia complex objects with their specific historicity.

For the database administrator the choice of whether to put the dynamicity at the attribute or entity levels of whether at the level of sub-objects or at the level of snapshots is a design problem which is application-dependent. Future DBMS's, however, must provide their users with definition and manipulation facilities for object versions [Bancilhon 88, Banerjee 87a, Dittrich 85].

Another important problem is related to dynamic changes of the database schema. Of course we can consider that schema information corresponds to special kinds of objects and provide history for them but this leads to non trivial problems of object compatibility [Banerjee 87b]. We return to this issue below.

Let us now consider complex objects as they are defined by [Bancilhon 85] or others [Abiteboul 86, Abiteboul 87, Adiba 88, Schek 86, Stonebraker 86, Valduriez 85]. Objects are recursively defined as follows: (1) Elementary domains, integer, real, string are atomic objects; (2) If O1, O2,...On are objects, and A1, A2,... An are distinct attribute names, then [A1:O1, A2:O2,...An:On] is a tuple object; and (3) If O1, O2, ...On are objects, then {O1, O2, ...On} is a set object.

In these objects we have a distinction between values and types. For instance if we consider database= [scientist: {[name, age, picture, education: {[degree, year, university]}, member: {organization}]}, this defines the type of MCO scientist which constitutes the database. MCO scientist gives for each occurrence name, age, picture, education as a set of degrees and membership as a set of professional organizations.

Applying historical concepts to this approach is not straitghforward. For instance an atomic object cannot be historical because it is effectively a constant. An attribute value in a tuple object can be updated and also it is possible to change the content of a set object, hence giving different versions of complex objects. Here, we have considered that type (or schema) is invariant., so we can apply historical notions to the component objects but also to the root object. However the semantics of linking objects and histories is not clear and requires more developments.

In an object-oriented approach, we have first to consider that an object has a unique identifier but also to consider types and methods [Lecluse 87, Banerjee 87a]. Here again, we have to characterize how to apply historical notions to all these elements. A rather natural starting point is to consider that once created, an object has an invariant identifier. Value is going to change and here again we have to differentiate between invariant types and non invariant types.

In order to solve the problem, we are currently defining a model for versions and histories in an object oriented approach. We consider the following basic notions:

* Each object is a 4-tuple *(id, type, value, method)* where *id* denotes the (invariant) identifier of the object, *type* corresponds to the schema (similar to the complex object model), *value* is the value of the object and *method* is the set of procedures associated to the object. We did not use the class concept yet in order not to introduce confusion. However, we consider that a given object is an instance of a class which gathers all the objects of a given type. Classes are of course organized in a hierarchy or a lattice.

* Historical objects are characterized by *(id, H(type), H(value), H(method))* where H means that we consider different states of the corresponding element. We are going to discuss only H(value) and consider that H(type) = type which means that type is invariant and the same for method (see [Kim 88] for discussion about schema evolution). Discussion

of the problem in all its generality is delayed to another paper. So, we consider *(id, H(value))* for an object. We have *H(value) = [HV, CV]* where HV is the historical value and CV the current value.

* We make the distinction between linear histories and tree structured histories. In linear histories, we have a sequence of (ti, vi) where the tis refer to a specific "calendar" (or clock) where values of the objects are characterized.: at ti, the object identified by id has value vi. In this case CV is simply the current value of the object.

In tree structured histories we consider that from a given object version v1 we can derive **several alternative versions** v11, v12, ... v1n. In this case, HV is a tree. The root is the first version of the object, and leaves represent the current versions of the object. For each of them, the path from the root is a linear history. CV is the set of all the current versions i.e. the set of leaves.

One important problem is to consider that an object together with its history remains an object as a whole or can be viewed as a current version and, separately, as a history. References to objects or to parts of them increase the complexity. Hence the identifier is an insufficient reference to the object if we want to refer to a specific version.

This model is very general and should be improved. We use it as a strarting point for defining histories and versions in applications such as CAD or CASE and we are currently trying to develop it in collaboration with people of the O2 model [Lecluse 87, Bancilhon 88].

CONCLUSIONS

Several research issues are open; more work is needed for histories and versions in the framework of object-oriented databases which seems to be a very promising field. Histories for objects should refer to the evolution of types (or classes), and methods. Also complex object semantics related to histories and versions should be studied more carefully and experiments performed in order to built version servers or specific mechanisms to help the user defining and manipulating his/her histories.

An important problem is that it is not sufficient to record successive or alternative versions of objects but also: (1) to be able to decide if such an evolution is valid according to specific rules which define object behaviour and (2) to record the actions which trigger such an evolution, creating specific link between different versions.

REFERENCES

[Abiteboul 86] S.ABITEBOUL, N.BIDOIT
"Non first Normal form relations: an algebra allowing data restructuring"
Journal of Computer and System Science, December 1986

[Abiteboul 87] S.ABITEBOUL, S.GRUMBACH
"Bases de données et objets structurés"
Technique et Science Informatique, Vol 6, N° 5, Nov. 1987

[Adiba 80] M.ADIBA, B.LINDSAY
"Database snapshots"
Proceedings of 6th VLBD Montreal,Canada, Oct 1980

[Adiba 81] M.ADIBA
"Derived relation : A unified mechanism for views,snapshots and distributed data"
Proceedings of 7th VLBD Cannes, France, Sept 1981

[Adiba 85] M.ADIBA, N.BUI QUANG NGOC, J.PALAZZO
"Time concepts for generalized data bases"
ACM Annual Conference, Denver Colorado, USA, Oct 1985

[Adiba 86] M.ADIBA, BUI QUANG NGOC
"Historical multimedia databases"
VLDB 1986, Kyoto (Japan)

[Adiba 87a] M.ADIBA, BUI QUANG NGOC
"Dynamic database snapshots, albums and movies"
TAIS: Temporal Aspects in Information Systems, France, May 1987

[Adiba 87b] M.ADIBA, BUI QUANG NGOC, C.COLLET
"Aspects temporels, historiques et dynamiques des bases de données"
Technique et Science Informatique, Vol 6, N°5, Nov 1987

[Adiba 88] M.ADIBA, C.COLLET
"Management of complex objects as dynamic forms"
VLDB 1988, Los Angeles (USA)

[Anderson 81] L.ANDERSON
"The Database semantics of time"
Doctoral Thesis, University of Washington, Jan 1981

[Anderson 82] L.ANDERSON
"Modelling Time at the conceptual level"
Proceedings of International Conference on DB, Academic Press , Israel 1982

[Bancilhon 85] F.BANCILHON, S.KHOSHAFIAN
"A calculus for complex objects"
MCC Technical Report DB-110-85, Octobre 1985

[Bancilhon 88] F.BANCILHON
"Object-Oriented Database Systems"
Rapport Technique Altaïr 16-88, Janvier 88

[Banerjee 87a] J.BANERJEE, H.T.CHOU, J.GARZA, W.KIM
"Data model issues for Object Oriented applications"
ACM TOIS, Vol 5 Num 1, January 1987

[Banerjee 87b] J.BANERJEE, W.KIM, H.J. KIM, H.F.KORTH
"Semantics and implementation of schema evolution in Object-Oriented databases"
ACM SIGMOD, 1987

[Bolour 82] A.BOLOUR, L.ANDERSON, J.DEKEYSER; T.WONG
"The role of Time in Information processing: A survey"
ACM SIGMOD Record 12.3 April 1982

[Bui 86] N.BUI QUANG
"Dynamic aspects and time management in generalized database systems"
Thesis PhD, INPG, Grenoble France, Nov 1986 (in french)

[Clifford 83] J.CLIFFORD, D.S.WARREN
"Formal semantics for time in database"
ACM TODS June 1983 Vol 8 N 2

[Dadam 84] P.DADAM, V.LUM, H.D.WERNER
"Integration of Time versions into a Relational Databases System"
Proceedings of 10th VLDB, Singapour, Aug 1984

[Dittrich 85] K.R.DITTRICH, R.A.LORIE
"Version support for engineering database systems"
IBM Research Report RJ 4769 July 1985

[Gadia 85] S.K.GADIA, J.H.VAISHNAV
"A Query Language for A Homogeneous Temporal Database"
Proceedings of Conf on Principles of Database Systems, Apr 1985

[Katz 83] R.H.KATZ, T.J.LEHMAN
"Database support for versions and alternatives"
Research Report University of California Berkeley, 1983

[Katz 86] R.KATZ, E.CHANG, R.BHATEJA
"Version modeling concepts for CAD databases."
ACM SIGMOD Washington DC, june 1986

[Kemper 87] A.KEMPER, P.C.LOCKEMANN, M.WALLRATH

"An Object-Oriented Database system for engineering applications"
SIGMOD 1987

[Kim 88] W.KIM, H.T.CHOU
"Versions of schema for object-oriented databases"
VLDB Conference, Los Angeles, August 1988

[Klopproge 81] M.R.KLOPPROGE
"TERM : An approach to include the Time dimension in the Entity-Relationship Model"
Proceeding 2nd Inter Conf on E-R Approach Washington 1981

[Koenig 81] S.KOENIG, R.PAIGE
"A Transformational framework for The Automatic Control of Derived Data"
Proceedings of 7th VLDB, Cannes France, Sept 1981

[Lindsay 86] B.LINDSAY, L.HAAS, C.MOHAN, H.PIRAHESH, P.WILMS
"A snapshot differential refresh algorithm"
Proceedings of ACM-SIGMOD Washington DC, May 1986

[Lécluse 87] C.LECLUSE, P.RICHARD, F.VELEZ
"O2, an Object Oriented Data Model"
Rapport Technique Altaïr 10-87, Septembre 1987

[Lum 84] V.LUM et all.
"Designing DBMS support for the Temporal dimension"
Proceedings of ACM SIGMOD Conference, June 1984

[McKenzie 86] E.McKENZIE
"Bibliography : Temporal Databases"
ACM SIGMOD RECORD Vol 15 N 4 Dec1986

[Navathe 87] S.B.NAVATHE, R.AHMED
"TSQL: A language Interface for History databases"
Temporal Aspects in Information Systems North Holland 1987

[Schek 86] H.J.SCHEK, M.H.SCHOLL
"The relational model with relation-valued attributes"
Information Systems Vol 11, N° 2, 1986

[Shoshani 86] A.SHOSHANI, K.KAWAGOE
"Temporal data management"
VLDB Kyoto (Japan) August 1986

[Snodgrass 85] R.SNODGRASS, I.AHN
"A Taxonomy of Time in Databases"
Proceedings of ACM SIGMOD May 1985

[Snodgrass 86] R.SNODGRASS
"Research concerning time in Databases : project Summaries"
ACM SIGMOD RECORD Vol 15 N 4 December 1986

[Snodgrass 87] R.SNODGRASS
"The Temporal Query Language TQuel"
ACM TODS, Vol 12, N°2, June 1987, p.247-298

[Stonebraker 86] M.STONEBRAKER, L.ROWE (Editors)
"The POSTGRES papers"
Memorandum No. UCB/ERL M86/85, November 1986

[Tsichritzis 87] D.TSICHRITZIS ed.
"Objects and Things"
Technical Report on Object Oriented System,
Centre Universitaire d'Informatique, Université de Genève, 1987

[Valduriez 85] P.VALDURIEZ, S.KHOSHAFIAN, G.COPELAND
"Implementation techniques for complex objects"
MCC Research Report, 1985

[Velez 85] F.VELEZ
"LAMBDA: An entity-relationship based language for the retrieval of structured documents"
ER conference, 1986

Objects in Time

James Clifford and Albert Croker

Information Systems Area
Graduate School of Business Administration
New York University
New York, New York 10006

Abstract

Two recent lines of database research, proceeding independently, have been concerned with providing a richer, more intuitive view of information at the user level. Historical database research has focused on ways to provide users with a view of information anchored and evolving in the temporal dimension. Object-oriented database research focuses on encapsulating both the structure and the behavior of the objects that users intend to model. In this paper we explore how these two lines of research might be brought together, providing to the user the representation and management of *objects in time.*

1. Introduction

Various proposals have been made for incorporating a temporal component into a database system [BADW82,McK86,TAI87]. Usually these proposals have been defined as extensions to the relational data model. In this paper we discuss the modeling of historical data in the context of an object-oriented data model.

Most object-oriented systems (see, for example, [Dit86] and [MSOP86]) owe their origins to the programming language *Smalltalk* [GR83]. Objects, the basic data constructs used in these systems, have proven to be both a powerful and flexible modeling construct. The power of objects arise in part from their ability to encapsulate both structure and behavior. The flexibility with which objects can be used as modeling constructs is due to the sets of data types that can be combined to define objects, the ability to nest the structure of objects, and the ability to encapsulate operations or *methods* on these objects in the manner of abstract data types. Objects are defined as consisting of values that are themselves objects; this nesting terminates with a set of **primitive objects**, such as integers, reals, and characters, that are built into the system. Thus an object that is used to represent an *employee* entity can include a component, say *salary*, whose value is an object representing the salary of that employee.

Object-oriented databases, like other types of databases, are used to model some aspect of the world. Each relevant entity and relationship in the modeled world is represented as an object in the database. Over time the various entities and relationships modeled by

the system may change. For example, when modelling employee entities, it is likely that employees may change departments, and that their salaries can be expected to change from time to time.

The traditional view of a database is that its state reflects that of the world at some specific, real or imaginary, instance of time; this instance being determined by the last update to the database. With each update the previous state of the database is lost. In contrast to this view, the state of a historical database models the world as it exists and has existed over some specified period of time [CW83]. If the equating of database objects with entities and relationships is to be retained in the context of an object-oriented database, then it is necessary for these objects to model the evolution of these real world objects over time.

In this paper we show how database objects can be defined in such a way that they can be viewed meaningfully in the context of a historical database. We call objects that are defined in this way **historical objects**. In addition to defining historical objects we also address various issues relating to their use in representing historical data. We do not present a formal model for integrating a treatment of time with a treatment of data as objects; to do so would be premature. Rather we discuss, from an intuitive point of view, those temporal aspects and properties which we believe ought to be captured by any system intended to represent our intuitive notions of "objects" and how they exist in time.

In the remainder of this paper we define what we mean by a historical object, and discuss several issues related both to the structure and to the manipulation of such historical objects.

2. Historical Objects

2.1. Introduction

Like the entities and relationships that they model, an object is characterized by some set of properties. We shall refer to these properties as **attributes.**[1] For example, if an object $\mathcal{O}$ corresponds to some employee, say Karen, then the attributes of $\mathcal{O}$: *NAME*, *SALARY*, *DEPT*, and *MGR* correspond to similarly named properties of the entity that is Karen.

The notion of a key is not inherent to objects. It is possible for two objects of the same type to denote the same values for their corresponding attributes. In lieu of the standard notion of a key, objects are distinguished by an **essence.** (We discuss object essences in the next section.)

Under the traditional view of a database an attribute of an object denotes a single value, that is for consistency also viewed as an object. The attribute *SALARY* in the object representing the employee Karen denotes what we will assume to be Karen's current salary.

[1] The equivalent term used in *Smalltalk* is **instance variable**.

The other attributes in this object are interpreted similarly.

However, in the context of a historical database, if, for example, Karen has been employed and therefore relevant to the world being modeled by the database since "*January 1, 1980*", then it becomes reasonable to query the database about her salary on any day during her employment. Thus, unless some specific instance in time is understood or otherwise inferred, the denotation of "Karen's Salary" can be viewed as being not a single value, but all of her salaries during the time she was employed, that is, her salary history.

In order to accommodate this view we define a **historical object** as an object whose attributes denote functional values. These functions, which, again for consistency, are themselves objects, define a correspondence between objects of type *time* to objects of the appropriate type, for example objects of type *salary*, *department*, or *name*.

Often an entity or relationship is relevant to a database for only some restricted period of time. The period of time for which a historical object models an entity or relationship is called the **lifespan** of that object. The domain of the function denoted by an attribute of an object is restricted to exactly those times in the lifespan of that object. (If modifications to a database scheme are to be allowed then it may be desirable to change – for instance, to extend – the definition of an object lifespan. However, this topic is beyond the scope of this paper. [CC87] presents an extended relational model with tuple and schema lifespans.)

2.2. Object Identity

A major issue when dealing with objects is the issue of object identity – how are we to distinguish different objects. Indeed this issue is not a new one. In earlier data models the notion of a *key* was used to so distinguish different records or tuples. In logic the issue of the "essence" of something addresses the same idea – what property of an object is essential to its being itself, so that anything with that property must be that thing, and anything without that property cannot be that thing. (The issues of object identity, existence, cross-world identification, and object counterparts have a long history in the philosophical literature, e.g. [Lew68,Mon74b,Kri80].)

Chen and Warren [CW88] are specifically concerned with this issue; our approach differs considerably from theirs by the introduction of the notion of an *essence*. We believe that each object must have an *essence*, which is a time-invariant identifier shared by no other object. One refers to an object by means of its essence. If two essences are equal then by definition they refer to the same object. Component properties of objects – such as an attribute *SALARY*– have as their value functions from time objects (referred to by their essence) to some other type of object (referred to by its essence). They are, in the terminology of logic, *intensions*. ([Mon74a,Gal75]). The essence of these functions, for example a *SALARY* function, are in general not directly known to the user – instead they would be referred to indirectly as the value of some property of a more essential object, say *Karen*, whose essence the user would know.

Now the issue addressed by Chen and Warren is how to tell whether two partially-specified intensions are the same. We would contend that two intensions are the same if and only if they are the value of some nonessential property of the same essential object; otherwise, even though extensionally they may be equal (i.e., have the same value for every time) they are not equal. However if these two intensions are created as different by the user, they would have two different *essences* and thus, though as functions they might be extensionally equal, as objects they are not the same. Thus John's *SALARY* is *never* Mary's *SALARY*, though they might always be earning the same money.

These issues motivate the following definitions:

1. It is essential that a system be able to maintain the integrity of object identity. Since user-defined keys are notoriously *not* time-invariant, for example, even people's social security numbers have had to be changed, our system will need to create and manage time-invariant object identifiers. We call such an identifier an **essence**.

2. Equally relevant to the management of objects over time is the maintenance of *when* that object existed. We call this information the **lifespan** of the object. Since an object may have temporally disjoint periods of existence, a lifespan consists of a set of disjoint intervals of time; such an interval is called an **incarnation**.

3. If E is an essence, we denote the lifespan of E as $E.l$.

4. The essence of each primitive object is simply its name, and the lifespan of each primitive object is $\{[-\infty, +\infty]\}$.

2.3. Object Structure

Various proposals for representing data as objects have incorporated different constructors for defining complex object types (or *classes*) from the primitive types. Common examples of these constructors are *record* construction and *set* (or collection) construction. Without examining any particular such constructor, let us assume that some complex object type O is defined in terms of n simpler object types. I.e.,

$$O = O_1 + O_2 + \ldots + O_n$$

where the symbol "+" is to be interpreted generically as any such constructor. When instantiated, a complex object has its own distinct essence. Any of the operations on objects can of course be applied to complex objects.

Note, however, that when a new object E of type O is created, consisting of the n component objects $E_1, E_2, \ldots, E_n$ of types $O_1, O_2, \ldots, O_n$, respectively, the constraint

$$\forall i[E.l \subseteq E_i.l]$$

must always be satisfied; an object can exist only while its components exist.

Relationships, too, are complex objects. However objects can certainly exist in time independently of the relationships they form. Thus the following restriction should be imposed upon relationships:

$$E.l \subseteq E_1.l \cap E_2.l \cap \ldots \cap E_n.l$$

Relationships pose another interesting situation where it is perhaps best to leave the choice of representation up to the user. Consider, for example, the two marriages of Elizabeth Taylor and Richard Burton. Are these two distinct objects (with, therefore, two essences) or are they two incarnations of the same marriage? Either representation should be possible.

Note that as a consequence of this, the following holds:

$$[E_\alpha = E_1 + E_2 + \ldots + E_n] \wedge [E_\beta = E_1 + E_2 + \ldots + E_n] \not\rightarrow [E_\alpha = E_\beta]$$

i.e., not only can there be different relationships defined in terms of the same underlying objects, but there can even be different instances of the same relationship between the same objects.

3. Manipulating Historical Objects

The entity or relationship modeled by an object is assumed to be relevant to the database during certain periods of time. These time periods are reflected in the incarnations of the object's lifespan. Each incarnation begins with a time when the object becomes newly "existing" from the perspective of the application, and terminates with the execution of an operation that "kills" that existence.

An object is brought into being with the operation[2] **CREATE**. The method used to define the **CREATE** operation, if so written, could also define the initial values of the attributes of the instantiated object. **CREATE**(X, B) returns a new essence E which uniquely identifies a new object of type X; the lifespan $E.l$ is $\{[B, \textbf{now})\}$. Moreover, if X is a compound object type, say $X = X_1 + X_2 + \ldots + X_n$ then n essences $E_1 + E_2 + \ldots + E_n$ are also generated with the same lifespan as $E.l$.

The execution of a **KILL** operation, implemented with the appropriate object method, terminates the most recently opened incarnation in the referenced object's lifespan. **KILL(E,D)** finds the object with essence E and, assuming it has lifespan $\{[B_1, D_1], [B_2, D_2], \ldots, [B_n, \textbf{now})\}$, updates its lifespan to $\{[B_1, D_1], [B_2, D_2], \ldots, [B_n, D]\}$. If E is a compound object of type X and $X = X_1 + X_2 + \ldots + X_n$ then the lifespans of the n essences $E_1 + E_2 + \ldots + E_n$ are also updated.

After an object has been **KILL**ed (but not removed) from a database it may necessary to **REINCARNATE** it. For example, an employee may subsequently be rehired. The affect of a reincarnation of an object is to extend its lifespan by beginning a new

[2]The term used in Smalltalk for an operation is **message**.

incarnation. (This incarnation will be terminated by the next subsequent **KILL** operation that is executed on the object.) Only objects that have been previously **CREATE**d can be **REINCARNATE**d. **REINCARNATE**(E, B) finds the object with essence E and, assuming it has lifespan $\{[B_1, D_1], [B_2, D_2], \ldots, [B_n, D]\}$, updates its lifespan to $\{[B_1, D_1], [B_2, D_2], \ldots, [B_n, D_n], [B, \textbf{now})\}$. If E is a compound object of type X and $X = X_1 + X_2 + \ldots + X_n$ then the lifespans of the n essences $E_1 + E_2 + \ldots + E_n$ are also updated.

It may sometimes be necessary to merge or **IDENTIFY** two descriptions into one because two supposedly distinct objects are now realized to be in fact the same object. **IDENTIFY**(E_1, E_2) finds the objects with essences E_1 and E_2 and, assuming that the object descriptions are "compatible"[3], creates a new object with the merged descriptions of E_1 and E_2 and gives it the essence E_1; the essence E_2 is no longer useable except as an alias for E_1;

It may be necessary, though perhaps forbidden in certain highly sensitive applications, to delete or **DESTROY** permanently any trace of an object from the system. **DESTROY**(E) finds the object with essence E and, removes it from the system. The essence E is thereafter and forever unusable. Once **DESTROY**ed an object cannot be **REINCARNATE**d.

Access to the attributes of historical objects is achieved in the conventional way, through the specification and invocation of the appropriate method. However, because of the structure of historical objects the value of an accessed attribute may have to be manipulated further.

Assume that the expression $\mathcal{O}$ **A** is used to invoke the method that retrieves the value denoted by attribute **A** of object $\mathcal{O}$. When the method invoked is that of an historical object then the object that is accessed is a function. For example, if **KAREN** is the name of the historical object modeling the employee Karen, then **KAREN salary** returns the function that represents Karen's salary history, and thus associates a salary with each time in the lifespan of **Karen**.

Since the functions denoted by the attributes of historical objects are themselves objects they can, and do, have methods associated with them. In particular, we assume that each such function object $\mathcal{F}$ includes a method that when invoked by the expression $\mathcal{F}$ **at:** *time* returns the value that $\mathcal{F}$ associates with the time denoted by *time*. The expression **KAREN salary at:** "March 1, 1981" returns the value of Karen's salary on the specified date.

Similarly attribute updates are accomplished using an expression of the form $\mathcal{O}$ **A at:** *time* **put:** *value*. This expression updates object $\mathcal{O}$ by extending the function denoted by attribute **A** so that it associates with *time* the value specified by *value*.

[3]What this means is an issue in its own right; see [CC87] for further details on this issue.

4. Conclusion

We believe that it is inconceivable to successfully develop an object-oriented model of data without providing for the modelling of the temporal dimension of objects. This paper represents a modest beginning toward amalgamating these two lines of research, object-orientation and historical data modelling, thereby providing users with the ability to model *objects in time.*

References

[BADW82] A. Bolour, T. L. Anderson, L. J. Deketser, and H. K. T Wong. The role of time in information processing: a survey. *ACM SIGMOD Record*, 12(3):28–48, April 1982.

[CC87] J. Clifford and A. Croker. The historical relational data model (hrdm) and algebra based on lifespans. In *Proc. Third International Conference on Data Engineering*, pages 528–537, IEEE, Los Angeles, February 1987.

[CW83] J. Clifford and D.S. Warren. Formal semantics for time in databases. *ACM Trans. on Database Systems*, 6(2):214–254, June 1983.

[CW88] W. Chen and D.S. Warren. *Objects as Intensions.* Technical Report, Dept. of Computer Science, SUNY at Stony Brook, 1988.

[Dit86] *Proc. International Workshop on Object-Oriented Databases*, Pacific Grove, CA, September 1986.

[Gal75] D. Gallin. *Intensional and Higher-Order Modal Logic.* North-Holland, Amsterdam, 1975.

[GR83] A. Goldberg and D. Robson. *Smalltalk-80: The Language and its Implementation.* Addison-Wesley, Reading, MA, 1983.

[Kri80] S. Kripke. *Naming and Necessity.* Harvard University Press, Cambridge, MA, 1980.

[Lew68] D. Lewis. Counterpart theory and quantified modal logic. *The Journal of Philosophy*, 65(5):113–126, March 1968.

[McK86] E. McKenzie. Bibliography: temporal databases. *ACM SIGMOD Record*, 15(4):40–52, December 1986.

[Mon74a] R. Montague. *Formal Philosophy: Selected Papers of Richard Montague.* Yale University Press, New Haven, 1974.

[Mon74b] R. Montague. *On the Nature of Certain Philosophical Entities*, pages 148–187. Yale University Press, New Haven, 1974.

[MSOP86] D. Maier, J. Stein, A. Otis, and A. Purdy. Development of an object-oriented dbms. In *Proc. of the Conference on Object-Oriented Programming Systems, Languages and Applications*, pages 472–482, September-October 1986.

[TAI87] AFCET. *Temporal Aspects in Information Systems*, May 1987.

THE ROLE OF TEMPORAL ELEMENTS IN TEMPORAL DATABASES

Shashi K. Gadia
Computer Science Department
Iowa State University
Ames, IA 50011
(515) 294-4377

Abstract. The objective of a temporal database is to model changing aspects of objects. The main technique in a temporal database is to use timestamps to indicate the periods of validity for the values taken by an attribute of an object. We argue that instants or intervals of time are not suitable timestamps; instead, one should use finite unions of intervals, called *temporal elements*. Although a temporal element seems to be a complex data type, it leads to simpler user interfaces. The concept of a temporal element is easily generalized to more than one dimension.

1. INTRODUCTION.

Timestamping the changing values of an object is the main tool used for modeling of temporal information. One is naturally lead to the possibility of more than one type of time; and they have been catalogued in [SA]. An object has a history in the real world, and we say that such a history exists with respect to the *real world time* (*valid time* in the terminology of [SA]). Our changing knowledge of such a history is incorporated in the database through updates. A computer system has its own clock, which is used to model *transaction time*.

It is clear that the real world and transaction times play some role in the way databases are used in practice. A database assists an organization in making decisions. It is reasonable to assume that the model of reality is not perfect; it can contain only an approximate understanding of the real world, and that this understanding gets refined with time. As a database is not fully correct, an organization is likely to take some incorrect actions; therefore, it may like to keep all records, correct and incorrect, concerning some aspect of its business practice. The usual practice is to maintain a log of all updates. Such a log does not enjoy the same status as the on-line database system, which typically reflects only the current knowledge of currently active objects. Querying a current snapshot of reality is understood rather well; one may use a query language such as SQL for this purpose. Querying the rich content of an update log has not been fully exploited.

[Sn] views a database as a sequence $\{\mathscr{D}_t$: t is a transaction time instant$\}$, where $\mathscr{D}_t$ is our knowledge of the real world history at transaction time t. This model allows one to rollback to a previous knowledge of reality. In [Sn], the real world timestamps are viewed as intervals or as instants, and the transaction timestamps are viewed as instants.

After the introduction of temporal elements in [Ga,GV], a consensus seems to have emerged that their use is appropriate as the real world timestamps [CC,MS]. The problems arising from the use of intervals as real world timestamps have been enumerated in [GY1]. We briefly discuss some of these problems in Section 2. However, such a consensus does not exist for transaction time. We propose the following: (i) Temporal elements should be used as the transaction timestamps. (ii) Although, when the real world and the transaction time dimensions are put together, the former seems to be nested inside the latter, we should disregard this fact and use two-dimensional timestamps that are finite unions of rectangles. (A rectangle is the cross product of two intervals of time.) The advantage of this approach is that we obtain powerful and natural navigational paths to query the database. This is illustrated in Section 4.

This paper is organized as follows. In Section 2, we present a model for the real world time. In Section 3, we introduce transaction time, considering instants as transaction timestamps. In Section 4, we consider two dimensional temporal elements as the timestamps, revisit an example from Section 3 and show that it becomes much simpler.

The paper is written in an informal style to exhibit what lies at the heart of the problem of timestamping. The formal details are rather involved and covered in [GB,GY2], where the concept of a weak relation, introduced in [Ga], plays a central role.

2. THE REAL WORLD TIME DIMENSION.

We assume that a universe of time instants [0,NOW], where NOW denotes the current time, together with a linear order $\leq$, is given. The set theoretic operations of union, intersection, and complementation on timestamps correspond to the natural language constructs "or", "and", and "not". Intervals of time are not closed under intersection and complementation, hence they are not adequate as timestamps. A *temporal element* is a finite union of intervals in [0,NOW]. An interval in [0,NOW] is obviously a temporal element. An instant t may be regarded as a temporal element by identifying it with the interval [t,t]. We use the variables μ, ν, μ_1, ν_1, ... to denote temporal elements. The union and intersection of μ and ν are denoted as $\mu+\nu$ and $\mu*\nu$, respectively, and the complement of μ (with respect to the universe [0,NOW]) is denoted as $-\mu$. The set of all temporal elements is a Boolean algebra under +, *, and -, with $\emptyset$ and [0,NOW] as its minimum and maximum elements. We assume that [0,NOW] consists of instants 0, 1, 2, ..., NOW.

To capture time variant properties of objects, we introduce the notion of a temporal assignment [Ga]. A *temporal assignment* (or simply an *assignment*) ξ to an attribute A, with a temporal element μ as its *temporal domain*, is a function from μ, such that for each $t \in \mu$, $\xi(t)$ is an element of dom(A). If ξ is an assignment, then $[\![\xi]\!]$ denotes its temporal domain, $|\xi|$ denotes its range $\{\xi(t): t \in [\![\xi]\!]\}$, and $\xi \restriction \mu$ denotes the temporal assignment which is the restriction of ξ to $\mu * [\![\xi]\!]$, where μ is a temporal element. If ξ_1 and ξ_2 are assignments to A and B, respectively, and θ is an operator in dom(A)×dom(B), then we define $[\![\xi_1 \theta \xi_2]\!] = \{t \in [\![\xi_1]\!] * [\![\xi_2]\!]: \xi_1(t) \theta \xi_2(t) \text{ holds}\}$.

Example 1. Suppose A and B are attributes such that dom(A) = dom(B) = {a,b,c}. Assume that $<$ is a transitive relation on dom(A)×dom(B) satisfying a<b<c. Suppose $\xi_1 = \langle [0,5]+[9,9] \mapsto c, [6,7] \mapsto a \rangle$ and $\xi_2 = \langle [3,9] \mapsto a \rangle$ are assignments to A and B, respectively. Then $\xi_1 \restriction [7,10] = \langle [7,7] \mapsto a, [9,9] \mapsto c \rangle$, $[\![\xi_1 < \xi_2]\!] = \emptyset$, and $[\![\xi_2 < \xi_1]\!] = [3,5] + [9,9]$.

Tuples, relations, and databases. A *tuple* τ over a scheme R is a function from R, such that for each $A \in R$, $\tau(A)$ is an assignment to A. If τ is a tuple over R, and μ is a temporal element, then $\tau \restriction \mu$ is the function from R, such that $(\tau \restriction \mu)(A) = \tau(A) \restriction \mu$, for every $A \in R$. If τ is a tuple over R, such that $\tau(A)$ is empty for some $A \in R$, then we say that τ is *null*. A *relation* over a scheme R is a finite set of non-null tuples over R. The temporal domain of a relation r over R is defined as $[\![r]\!] = +_{\tau \in r, A \in R} [\![\tau(A)]\!]$. A *database* is a finite set of relations.

Key. Suppose r is a relation over R. We say that $K \subseteq R$ is the key of R, if (i) $|\tau(A)|$ is a singleton for every $A \in K$, and (ii) If τ and τ' are tuples of R, then $\forall A \in K(|\tau(A)| = |\tau'(A)|)$ if and only if $\tau = \tau'$.

Example 2. A database consisting of an emp relation over NAME SALARY DEPT with NAME as its key, and a management relation over DEPT MANAGER with DEPT as its key is shown in Figure 2.1. Note that [[emp]] = [0,NOW] and [[management]] = [11,49] + [71,NOW].

The relational algebra. The algebra consists of three kinds of expressions: temporal expressions, Boolean expressions, and relational expressions.

Temporal expressions are formed using $[\![\xi_1 \theta \xi_2]\!]$, $[\![r]\!]$, and the set theoretic operators +, * and -. The following example illustrates a temporal expression.

Example 3. Consider the database of Figure 2.1. Then [[SALARY $\neq$ 25K]] + [[DEPT = Toys]] is a temporal expression, and we denote it as μ. Suppose τ is John's tuple in the emp relation. Then SALARY(τ), the result of substituting τ in SALARY, evaluates to $\langle [11,49] \mapsto 15K, [50,54] \mapsto 20K, [55,60] \mapsto 25K \rangle$. Now, [[SALARY $\neq$ 25K]](τ) evaluates to [11,49] + [50,54] = [11,54]. Similarly, [[DEPT = Toys]](τ) = [11,44]. Thus $\mu(\tau)$ = [11,54] + [11,44] = [11,54]. If τ is Tom's tuple, then $\mu(\tau)$ = [0,20] + [41,51]. If τ is the tuple of Mary or Inga, $\mu(\tau) = \emptyset$. If τ is Leu's tuple, then $\mu(\tau)$ = [31,NOW].

NAME	SALARY	DEPT
[11,60] John	[11,49] 15K [50,54] 20K [55,60] 25K	[11,44] Toys [45,60] Shoes
[0,20]+[41,51] Tom	[0,20] 20K [41,51] 30K	[0,20] Hardware [41,51] Clothing
[71,NOW] Inga	[71,NOW] 25K	[71,NOW] Clothing
[31,NOW] Leu	[31,NOW] 23K	[31,NOW] Toys
[0,44]+[50,NOW] Mary	[0,44]+ 25K [50,NOW]	[0,44]+ Credit [50,NOW]

The emp relation

DEPT	MANAGER
[11,49] Toys	[11,44] John [45,49] Leu
[41,47]+[71,NOW] Clothing	[41,47] Tom [71,NOW] Inga

The management relation.

Figure 2.1. The personnel database.

Boolean expressions are formed using the constants TRUE and FALSE, $\mu \subseteq \nu$, $\mu = \nu$, where μ and ν are temporal expressions, $r = \emptyset$, where r is a relational expressions, and the Boolean operators $\vee$, $\wedge$ and $\neg$.

The relational expressions are $r \cup s$, $r - s$, $r \cap s$, $\Pi_X(r)$, $r \times s$, renaming operator $\rho_{A \leftarrow B}(r)$, which renames A to B, and $\sigma(r; f; \mu)$, where f is a Boolean expression and μ a temporal expression. The selection operator $\sigma(r; f; \mu)$ is interesting: it stands for $\{\tau \mid \mu(\tau)$: $\tau \in r$ and $f(\tau)$ holds$\}$. We also define $r[f; \mu]s = \{\tau_1 \circ \tau_2 \mid \mu(\tau_1 \circ \tau_2)$: $\tau_1 \in r$, $\tau_2 \in s$ and $f(\tau_1 \circ \tau_2)\}$ holds$\}$, where $\tau_1 \circ \tau_2$ denotes the concatenation of τ_1 and τ_2. Each occurrence of NOW should be replaced by its current value at the query execution time. The semantics of each relational expression should determine the key of the resulting relation [GY2,GB].

Example 4. List the starting salaries and departments of the employees who are currently employed by the organization. The query is expressed as σ(emp; NOW $\subseteq$ [[DEPT]]; fi([[SALARY]])}, where the operator fi(μ) computes the first instant in μ. We remark that this query would become more complex if intervals are used as timestamps instead of

temporal elements. This is because for a given employee, the first instant of his/her employment may be contained in a tuple which is different from the tuple in which the current information about that employee resides. If intervals are used as timestamps, we will have to, either, use some aggregate operation to collect all the information about an employee, or use a join.

Example 5. Give the information about all employees in Toys department during the time when John was a manager in some department. We may first compute μ = $[\![\sigma(\text{management; TRUE; } [\![\text{MANAGER=John}]\!])]\!]$ and then express the query as σ(emp; TRUE; $\mu * [\![\text{DEPT=Toys}]\!]$). Here, the use of the temporal expression μ, which evaluates to a temporal element, and not necessarily an interval, helps give a simple query and also reduces the need for optimization.

3. INSTANTS AS TRANSACTION TIMESTAMPS.

In this section we attempt to model our changing knowledge of the history of the real world along the lines of [Sn]. We use instants as transaction timestamps. We call the resulting model the *evolution model*. We assume that [0,NOW] denotes the universe of the transaction time, and refer to an instant in [0,NOW] as a *state*.

If ξ is a assignment and t a transaction time instant, then $\xi' = \langle \xi,t \rangle$ is a *evolution assignment*, and state(ξ') is defined to be t. *Evolution tuples*, *evolution relations*, and *evolution databases* are defined in a natural manner. Note that the evolution model does not put the entire evolution of an object in a single tuple.

In querying a database, a user is expected to refer to an object by its state; to make this possible we introduce evolution terms. If A is an attribute, and μ a temporal element, then A_μ is an *evolution term*. At run time, $A_\mu(\langle \xi,t \rangle)$ evaluates to $A(\xi)$ if $t \in \mu$, and to Ω if $t \notin \mu$ (Ω means undefined). When we substitute $\langle \xi,t \rangle \theta \langle \eta,t' \rangle$ in $[\![A_\mu \theta B_\nu]\!]$, the result is Ω if $t \in \mu \wedge t' \in \nu$ does not hold, otherwise it is $[\![\xi \theta \eta]\!]$. The definition of temporal expressions, Boolean expressions and relational expressions given in the previous section are easily generalized to the evolution model. The semantics of operators should deal with Ω in a careful manner. For example, for a relation to be computed, we should exclude tuples with the state Ω in any of its attributes.

Our venture is not very interesting if we do not allow μ in A_μ to be captured in a variable and be used in other queries. To allow this, we introduce the construct «r(A)», which stands for $+_{\tau \in r}$state(τ(A)).

Example 6. When did John's department information exist in the database. We assume that emp' denotes the evolution of emp. We express the query as follows.

«σ(emp'; $\text{NAME}_{[0,\text{NOW}]}$ = John;)(DEPT)».

Example 7. Is Tom's information existing at t = 30 correct? We assume that the information in the database state NOW is correct. Note that some information about Tom may have been added after t = 30; the query implicitly assumes that we should disregard such information. This is achieved by restricting Tom's current information to [0,30] before any comparison with the information in state t=30 is made. As the query is long, we express it by the following program.

$r_1 = \Pi_{\text{NAME DEPT}}(\sigma(\text{emp}'; \text{NAME}_{30} = \text{Tom}; [0,\text{NOW}]))$
$r_2 = \Pi_{\text{NAME DEPT}}((\sigma(\text{emp}'; \text{NAME}_{\text{NOW}} = \text{Tom}; [0,\text{NOW}])) \upharpoonright [0,30])$
$r_3 = \rho_{\text{NAME}\leftarrow\text{NAME}'}(\rho_{\text{DEPT}\leftarrow\text{DEPT}'}(r_2))$ (renaming of attributes)
$r_4 = \Pi_{\text{DEPT DEPT}'}(r_1[\text{NAME}=\text{NAME}']r_3)$
$r_5 = \sigma(r_4; \text{TRUE}; [\![\text{DEPT}=\text{DEPT}']\!])$
$f = ([\![r_5]\!] = \emptyset)$.

4. TREATING TIME DIMENSIONS UNIFORMLY.

In this section we show how the model of the previous section can be improved by treating the two time dimensions uniformly. If I and J are intervals in the real world time and transaction time dimensions, respectively, then I×J is called a *rectangle.* A 2-temporal element is a finite union of rectangles. A 2-temporal assignment to A with a 2-temporal element μ as its temporal domain, is a function from μ into dom(A). *2-tuples*, *2-relations* and *2-databases* are defined in a natural manner. This model allows us to put the whole evolution of an object into a single tuple. The algebraic operators are also defined in such a way that the history of one object is not fragmented across tuples [GY2,GB]. Now we revisit Example 7.

Example 8. The query of Example 7 is expressed as follows. Clearly the new expression is much simpler.

$\sigma(\text{emp}; \text{NAME} = \text{Tom}; [\![\text{DEPT} \upharpoonright \{30\}\times[0,30] = \text{DEPT} \upharpoonright \{\text{NOW}\}\times[0,30]]\!]) = \emptyset$.

5. SOME FINAL REMARKS.

[GB] gives a formal definition of an update log, and shows that it is resident in the model presented in Section 4. Therefore, an explicit update-log is redundant. This means that no update activity is discarded by our model, and therefore, it is self contained. Such a model is very rich in content; for example, it would allow querying of errors and updates, eliminating a need to deal with such problems manually. This is a promising application of temporal databases to mainstream databases. We feel that the database systems of the future will put some of the ideas generated by researchers in temporal databases to work.

ACKNOWLEDGEMENTS. This work was supported, in part, by the National Science Foundation grant #IRI-8810704. I like to thank Gautam Bhargava and Richard Snodgrass for their helpful suggestions.

REFERENCES

[CC] Clifford, James and Croker, Albert. *The Historical Relational Data Model (HRDM) and Algebra Based Upon Lifespans.* Third IEEE International Conference on Data Engineering, 1987, pp 528-537.

[Ga] Gadia, Shashi K. *A Homogeneous Relational Model and Query Languages for Temporal Databases.* To appear in December 1988 issue ACM-TODS.

[GV] Gadia, Shashi K. and Vaishnav, Jay. *A Query Language for a Homogeneous Temporal Database.* Proc. Fourth Annual ACM SIGACT-SIGMOD Symp. on Principles of Database Systems, March 1985, pp 51-56.

[GB] Gadia, Shashi K. and Bhargava, Gautam. *A Formal Treatment of Updates and Errors in a Relational Database.* Under Preparation.

[GY1] Gadia, Shashi K. and Yeung, Chuen-Sing. *Inadequacy of Interval Timestamps in Temporal Databases.* To appear in Information Sciences.

[GY2] Gadia, Shashi K. and Yeung, Chuen-Sing. *A Generalized Model for a Relational Temporal Database.* Proc ACM-SIGMOD Int. Conf. on Management of Data, June 1988, pp. 251-257.

[MS] McKenzie, E. and Snodgrass, R. *Supporting Valid Time: An Historical Algebra.* Technical Report TR87-008, Aug 1987. Computer Science Department. University of North Carolina, Chapel Hill.

[Sn] Snodgrass, Richard. *The Temporal Query Language TQuel.* ACM Transactions on Database Systems, Vol 12, 1987, pp 247-298.

[SA] Snodgrass, Richard and Ahn, I. *A Taxonomy of Time in Databases.* Proc ACM-SIGMOD Int. Conf. on Management of Data, May 1985, pp. 246.

[Ta] Tansel, A.U. *Adding Time Dimension to Relational Model Model and Extending Relational Algebra.* Information Systems, 1986.

REQUIREMENTS SPECIFICATION FOR A TEMPORAL EXTENSION TO THE RELATIONAL MODEL

Nikos A. Lorentzos * and Roger G. Johnson

Department of Computer Science,
Birkbeck College, London University
Malet Street
London WC1E 7HX
United Kingdom

ABSTRACT

A set of properties is identified, which should be satisfied by a temporal extension to the relational model. During the process of identifying these properties it is initially shown that certain semantic problems arise if time is treated as a means of stamping data in temporal databases. To overcome the problems encountered, it is next shown that a more general extension to the relational model needs to be made to define a model in which a generic interval is supported as a primitive data type. This extension has many practical applications. Furthermore, since a time-interval is the only type of interval which is used in temporal databases, the management of temporal data is shown to be one of the applications of this general model.

1. INTRODUCTION

The concept of a *Temporal Database* has been frequently used to denote databases from which pieces of data representing states of the real world, other than the current state, are not deleted. Conventionally in such a database, data is 'stamped' with the time during which it is *valid* or *in effect*. An interesting piece of work to incorporate time in a database is presented in [CLI83]. However, the first serious attempts to define a temporal extension to the relational model for the management of temporal data appeared at the end of 1985, when three different models were proposed, one by Gadia in [GAD85] and two more by Clifford and Tansel in [CLI85]. Since then three more have been defined, by Ariav [ARI86], Navathe and Ahmed [NAV86] and McKenzie and Snodgrass [MCK87a].

Certain properties, which a temporal data model should satisfy, are presented in [CLI85] and [MCK87b]. A different approach is undertaken in this paper and certain properties, which a temporal extension to the relational model should satisfy, are identified. During the process of identifying these properties two major conclusions are drawn, which the authors believe to be novel. The first is that, in an extension to the relational model, data should not be time-stamped. The second is the need for an alternative extension, more general than a temporal extension. As a consequence, the management of temporal data is only one of the application areas of this general extension.

On leave. Permanent address: Pergamou 14, 184 50 Nikaia, Greece.

The above topics are discussed in sections 2, 3 and 4 of this paper. Section 5 is a reference to the *Interval-Extended Relational Model*, whose definition has been based on the conclusions of this paper. Some research topics are summarised in the last section.

For reasons of notational convenience in this paper, "CRM" is used to denote the Conventional Relational Model, in which first normal form is maintained. Any extension to manage temporal data, is termed "TCRM".

2. PROPERTIES OF A TEMPORAL EXTENSION TO THE RELATIONAL MODEL

All data models, including TCRMs, have limited representational capabilities. One example is Clifford's model [CLI85], in which time is represented as a point. Another example is Gadia's model [GAD85], in which time is represented as an interval. In the majority of the TCRMs currently defined, time is almost exclusively represented either as a time-point or as a time-interval. However, people have a dual perception of time. Sometimes, they refer, for example, to a year which they interpret as a time-point, as in the phrase *"John was born in 1952"*. In other instances they refer to a period of years which they interpret as a time-interval. A typical example is the phrase *"John was the president of company A during the period 1980-1986"*. Taking the above remarks into consideration, in conjunction with the fact that time is a factor of primary importance in a TCRM, it can be argued that if both time-points and time-intervals are not supported in a TCRM then the model's representational capabilities will be unduly limited. Therefore, the first property is the following.

(P1) *Both time-points and time-intervals need to be supported by a TCRM.*

It should be noted that a time-point, for example the year 1980, can be represented in a database as the time-interval [1980, 1981). However, the meaning of property (P1) is that, in a TCRM, time-intervals and time-points should be represented as intervals and points, respectively.

The second property relates to the operational capabilities of a TCRM. Generally, every data model has limited operational capabilities and a TCRM cannot be an exception to this rule. However, if both time-points and time-intervals are supported by a TCRM, then operations must also be defined to transform between these two representations of time, otherwise the TCRM's operational capabilities will be severely restricted. This observation leads to the second property:

(P2) *In a TCRM, at least two operations need to be defined, one operation F, to transform from time-points to time-intervals and another one U to perform the inverse task.*

A third property is the following.

(P3) *All the operations defined in a TCRM need to be closed.*

It is beyond the scope of this paper to justify this property. An explanation is provided in [LOR88c].

Since a TCRM is an extension of the CRM, it implies that the following property needs also to be satisfied.

(P4) *Every relation R of the CRM, can be mapped to a corresponding one, TR, in the TCRM.*

This means that every piece of data, which can be represented in the CRM, can also be represented, in an equivalent way, in the TCRM. Indeed, if (P4) is not satisfied then the TCRM cannot be an extension of the CRM.

However, generally in a TCRM, temporal data is stamped with the time during which it is valid. This fact, in conjunction with the necessity to satisfy (P4) may give rise to certain semantic difficulties, as it is shown by some examples in the next section.

3. PROBLEMS OF TIME-STAMPING

Example 1: Consider the CRM relation CONSTANTS(Symbol, Value), which is used to record all the universal constants. One tuple of it, could be (π, 3.14), which states that π is equal to 3.14. Suppose TCRM is a temporal extension to the CRM. If (P4) is satisfied then one of the following two cases has to be true for the TCRM.

(i) CONSTANTS is a valid relation in the TCRM.

It is observed that if this is true then a semantic difficulty arises. As stated is section 2, data in the TCRM is time-stamped, so CONSTANTS cannot be a valid TCRM relation.

(ii) CONSTANTS is not a valid TCRM relation but its contents can be recorded, in an equivalent way, in a relation TCONSTANTS in the TCRM.

If it is assumed that (ii) is true then the data recorded in TCONSTANTS must be stamped with time. However, in this case another semantic problem arises, that although the value of π is independent of time, it has to be time-stamped. This contradicts the semantic property of relation CONSTANTS, of being independent of time.

With this example, it has been shown that if time is treated as a stamp in a TCRM, then certain important semantic problems arise with the TCRM's representational capabilities. The next example shows that certain problems also arise with the TCRM's operational capabilities.

EMPLOYEE

Name	Salary	Manager
john	[d2, d6) 10k [d8, d12) 12k	[d2, d8) mark [d8, d14) mick
george	[d1, d12) 8k [d12, d16) 9k	[d1, d10) mick [d10, d16) mike

Figure 1: A temporal relation incorporating attribute time-stamping.

Example 2: Consider the relation in Figure 1. It is a relation of a TCRM proposed by Tansel in [CLI85]. Time-intervals are used to stamp the salary and managers of every employee for various periods. Consider also the query *"Retrieve the names of all the persons who have ever been managers."*

This is not a hypothetical, theoretical query, but a natural one, which could be issued in a real environment. A number of strategies could be adopted to obtain an answer. One solution is to project relation EMPLOYEE on attribute Manager. In this case, the resulting relation will consist of the names of the managers and the associated time-stamps. Clearly, this is not exactly the result intended by the query. In particular, the query specifies explicitly that the names of the managers need to be retrieved but the time-intervals, with which these names are stamped, are not required. Therefore, if a TCRM has not been supplied with operations to 'project out' the time-stamps then its operational capabilities are restricted.

Assume alternatively, that certain operations have been defined in a TCRM, which can 'project out' the time-stamps. Then R, the resulting relation, will contain data which is not stamped with time. In this case a semantic problem arises concerning the validity of R as a TCRM relation, since its data is not time-stamped.

Various approaches are now discussed to overcome the problems which have been presented in examples 1 and 2.

With respect to the first example, one possible solution could be for 3.14 to be stamped, in a TCRM relation, with the time-interval $(-\infty, +\infty)$. This would be an indication that π=3.14 *for all times*. This solution is not very satisfactory because if $(-\infty, +\infty)$ is projected out by the operation described in example 2, then the question will again arise whether the remaining data is still temporal or not.

As a second solution, let a convention be made that if data is not time-stamped then it is valid *for all times*. This option works perfectly for the first example, where CONSTANTS becomes a valid TCRM relation. Unfortunately, it is not completely satisfactory if it is applied to the second example. In particular, if the query in this example is answered, the result will be a relation consisting of the tuples (mark), (mick) and (mike) but the interpretation that these persons have been managers *for all times* contradicts reality.

Another option could be for non-time-stamped data to be interpreted as being valid *at some time*. This seems to be a better solution, although an argument could be made that temporal data is not treated uniformly, in the sense that time is explicitly recorded in some relations whereas in others it is implied. Finally, the next example shows that, generally, stamping data may create confusion.

CLIMBERS

Team	Top	Date
A	everest	[h0, h100) d1 [h100, h180) d2
B	kilimanjiro	[h0, h150) d1 [h150, h200) d2 [h200, h250) d3

Figure 2: A relation in which time seems to be stamped with height.

Example 3: Consider the relation in Figure 2. It is used to record how high teams of mountaineers climb a particular mountain, on a specific day. For example, in the first tuple it can be seen that team A climbed Everest and on day d1 they climbed from h0 to h100. In one interpretation it could be argued that d1 time-stamps [h0, h100). However, if CLIMBERS is compared with relation EMPLOYEE in Figure 1, then, in an analogous way, it could rather be argued that [h0, h100) stamps day d1.

A similar example could be given to show that time-points might also be interpreted as being stamped.

It should be noted that Clifford, in [CLI87], associates a lifespan with every relation, in order to overcome the problems caused by time-stamping. However, in the authors' opinion, the previous examples demonstrate that a more radical approach is required. For this purpose the following additional property can be adopted.

(P5) *In a TCRM, data should not be time-stamped.*

This means that time should not be treated as a means of stamping data but it should be recorded and manipulated like any other piece of conventional data. Hence, if (P5) is applied to the portion of the relation in Figure 1, which concerns the managers of the employees, this data can

EMPLOYEE

Name	Manager	Period
john	mark	[d2, d8)
john	mick	[d8, d14)
george	mick	[d1, d10)
george	mike	[d10, d16)

Figure 3: A relation in which time is represented as data.

equivalently be represented as shown in Figure 3, where time is recorded in a distinct attribute. Although this representation is identical to that proposed by Snodgrass in [SNO87], there is a difference in the semantics. That is, in [SNO87] it is explicitly assumed that a value for Period time-stamps a value for Manager whereas, in the approach proposed here, time is interpreted as pure data. Because of this, no distinction is made between temporal data (which is time-stamped) and non-temporal data. Consequently, an operation, to project out Period, does not give rise to a semantic question of the form *"What is the time during which the data in the resulting relation is in effect?"*.

4. GENERIC INTERVALS

The contents of the relation in Figure 2 imply that intervals other than time-intervals do make sense in a database. The next example shows that whether an interval is a time-interval or not is, in many cases, a matter of interpretation.

R

Name	A
john	1
john	2
john	3
john	7
john	8
george	5
george	6
george	7

(a)

S

Name	A
john	[1, 4)
john	[7, 9)
george	[5, 8)

(b)

Figure 4: Two valid relations in an extension of the CRM to support intervals.

Example 4: Consider relation R in Figure 4. Its scheme is R(Name=STRING_12, A=INTEGER). For a given day, R could be used to record the hours at which various persons worked in a high radiation laboratory. The integers recorded in A represent hours, that is, they can be interpreted as time-points. Therefore, R is a valid TCRM relation. Then, from (P2), F can be defined so that S=F[A](R) to result in the relation in Figure 4(b),

which is also a valid TCRM relation.

In another interpretation, the same relation, R, could be used to record the projects in which the employees of an enterprise are involved. Now the integers, recorded in attribute A, cannot be interpreted as time-points. However, since from (P3) F needs to be closed, S=F[A](R) should give again the relation in Figure 4(b).

In addition to the above remark, consider another relation, CAR(City, Number). One tuple of CAR could be (athens, AA5360), which states that the authorities of Athens can issue the car number AA5360. The underlying domain of attributes City and Athens are not time-points. They are not even numeric. Again, since F needs to be closed, it has to be so defined that, for example, F[Number](CAR) to result in a valid relation. At the same time, it is observed that intervals over arbitrary alphabetic and alphanumeric strings do make sense. For example, [abrial, freedman) could represent the range of names of all the telephone holders who have been recorded in the first volume of the telephone index. Similarly, [AB5000, AC9000) could represent the range of valid car numbers which the authorities of a specific city can issue.

The above discussion makes it explicit that time-intervals are only one of many plausible types of generic interval which could be used in a database. Furthermore, by extension from the concept of a generic interval, some values like 1, abrial, AB5000, could be called generic points. Incidentally, it is worth noting that only generic points can be supported by the CRM. Therefore, the CRM could be called the *Point Relational Model.*

The above discussion implies that properties (P1), (P2) and (P4) can be replaced by the following one.

(P6) *The CRM needs to be extended to a model, XRM, which supports a generic interval data type, in addition to the support of a generic point data type. At least two operations, F and U, need to be defined in the XRM, F to transform from generic points to generic intervals and U to perform the inverse task.*

Furthermore, since a time-interval is only a special type of interval then, with respect to temporal databases, if (P5) is replaced by

(P7) *In the XRM, data should not be time-stamped,*

and (P3) replaced by

(P8) *All the operations defined in the XRM need to be closed*

the conclusion is that the XRM can also be applied to temporal databases.

In summary, (P6), (P7) and (P8) are the properties which need to be satisfied by the XRM.

5. THE INTERVAL-EXTENDED RELATIONAL MODEL

The previous discussion was the analysis of the problem for the management of temporal data in an extension of the CRM. In [LOR88c], the synthetic approach has been used, to formalise the XRM. A generic interval is supported as a primitive data type and an algebra, consisting of seven operations, has been defined. The first five of them are the known operations of relational algebra. The other two have been called FOLD and UNFOLD and play the role of F and U in (P7). All the operations are closed. The model has been called the Interval-Extended Relational Model. It has been proved that the XRM is, in all respects, a proper superset of the CRM. Functional dependencies have been investigated and two normal forms have been defined. Finally, a portion of the XRM has been implemented.

With reference to the management of temporal data, two predecessors of the model are reported in [LOR87] and [LOR88d]. In the former it is shown how temporal data can be updated and how periodic temporal data can be manipulated. In the latter various examples are given, to show that the XRM has a rich expressive power. It is a general model, since temporal databases are only

one of its many application areas. In [LOR88b] it has been shown how one of these predecessors can be applied to Soil Information Systems. The management of spatial data is another of its applications. A formal description of the XRM has appeared in [LOR88a].

6. CONCLUSIONS

A set of properties have been identified which an extension of the relational model, for the management of temporal data, should satisfy. During the process of identifying them, it has been shown that the time-stamping of temporal data causes certain problems. It has also been shown that one effective solution, for overcoming them, is for the extension of the relational model to be general enough to include temporal databases as one of its application areas. Further research includes the definition of an Interval-Extended Calculus and the definition of an extension of SQL.

ACKNOWLEDGEMENT

The authors would like to thank Professor Richard Snodgrass, University of North Carolina, for his comments and helpful criticism on an earlier draft of this paper.

REFERENCES

[ARI86] G. Ariav: *A Temporally Oriented Data Model.* ACM/Transactions on Database Systems 11(4), 499-527 (1986).

[CLI83] J. Clifford and D. S. Warren: *Formal Semantics for Time in Databases.* ACM/Transactions on Database Systems 8(2), 214-254 (1983).

[CLI85] J. Clifford and A. U. Tansel: *On an Algebra for Historical Relational Databases: Two Views.* Proceedings of the ACM SIGMOD International Conference on Management of Data, Austin, Texas, 247-265 (1985).

[CLI87] J. Clifford and A. Croker: *The Historical Relational Data Model (HRDM) and Algebra Based on Lifespans.* Proceedings of the Third International Conference on Data Engineering, 528-537 (1987).

[GAD85] S. K. Gadia and J. Vaishnav: *A Query Language for a Homogeneous Temporal Database.* Proceedings of the Fourth ACM SIGACT-SIGMOD Symposium on Principles of Database Systems, 51-56 (1985).

[LOR87] N. A. Lorentzos and R. G. Johnson: *TRA: A Model for a Temporal Relational Algebra.* Proceedings of the IFIP TC 8/WG 8.1 Working Conference on Temporal Aspects in Information Systems, Sophia-Antipolis, France, 13-15 May 1987, in *Temporal Aspects in Information Systems,* Ed. C. Rolland, F. Bodart and M. Leonard, North-Holland, 203-215 (1988).

[LOR88a] N. A. Lorentzos and R. G. Johnson: *An Extension of the Relational Model to Support Generic Intervals.* Proceedings of the International Conference on Extending Database Technology, Venice, Italy, 13-15 March 1988, in *Advances in Database Technology-EDBT'88,* Ed. J. W. Schmidt, S. Ceri and M. Missikoff, Springler-Verlag, 528-542 (1988).

[LOR88b] N. A. Lorentzos and V. J. Kollias: *The Handling of Depth and Time Intervals in Soil Information Systems.* To appear in Computers and Geosciences.

[LOR88c] N. A. Lorentzos: *A Formal Extension of the Relational Model for the Representation and Manipulation of Generic Intervals.* PhD Thesis, Birkbeck College, London University, August 1988.

[LOR88d] N. A. Lorentzos and R. G. Johnson: *Extending Relational Algebra to Manipulate Temporal Data.* To appear in Information Systems 13(3) (1988).

[MCK87a] E. McKenzie and R. Snodgrass: *Supporting Valid Time: An Historical Algebra.* Technical Report TR87-008, Department of Computer Science, University of North Carolina (1987).

[MCK87b] E. McKenzie and R. Snodgrass: *An Evaluation of Historical Algebras.* Technical Report TR87-20, Department of Computer Science, University of North Carolina (1987).

[NAV86] S. B. Navathe and R. Ahmed: *A Temporal Relational Model and a Query Language.* Technical Report TR-85-16, Computer and Information Sciences Department, University of Florida (1986).

[SNO87] R. Snodgrass: *The Temporal Query Language TQUEL.* ACM/Transactions on Database Systems, 12(2), 247-298 (1987).

Temporal Aspects of Version Management

Shamkant B. Navathe
and
Rafi Ahmed

e-mail: sham@bikini.cis.ufl.edu; rafi@bikini.cis.ufl.edu
Database Systems Research and Development Center, E 470 CSE
University of Florida, Gainesville, FL 32611
904-335-8460

Abstract

This paper discusses the basic notions of versioning in design databases. It explores the temporal aspects of version control and the semantics of temporal relationships among version objects. Some representation schemes are outlined and a few open problems are identified.

Introduction

Version control is a significant issue in design databases, which are of general applicability to the design process in engineering and software systems design and other related fields.

The size and complexity of many design projects has rendered existing database tools and support inadequate. Database systems designed for commercial applications suffer from the fact that they normally hold only one valid view of the world. In design environments, however, a developer might be interested in pursuing a particular design along several possible paths simultaneously, and may later select one of the alternatives, or merger of alternatives, as the final design. Therefore, several valid representations of an object will simultaneously exist in the database. Furthermore, a traditional database system, as it handles only regular and structured data, will not be able to deal with design data efficiently, since there are large variations in the size and structure of such data. We shall be using object-oriented terminology in the following discussion. However, the issues raised here are also relevant to other data models.

Generally, design of an object starts with a high level description of some aspects of that object. A named collection of information describing an object at the type level is called a *template.* A *refinement* of an object is another object that contains a qualitatively or quantitively different information than the previous ones in the direction of the ultimate design goal. A *version* of an object is its refinement which contains all the information necessary to instantiate the object. The versioning concept in a design database is manifested in families of distinct objects, each containing one common generic object and one or more version objects.

The refinement of design objects may proceed along several axes [Co88]. The design may be refined in a top-down fashion structurally; it may also be refined along an "improvement axis" -- where the same objects are technically improved using new technologies.

A system must provide the facility for creating and maintaining a version graph to record the evolution history of versions from the initial design through successor and predecessor relationships. It should be possible for versions of an object to be referenced either explicitly (as a

specific reference using a qualified name or an internal identifier), or implicitly by dereferencing (also called generic reference) according to some prespecified or default criteria.

An important issue in the version management is the formulation of a generalized definition of *version*. Ideally, a system should provide control facility at both the template and instance levels. Versioning at instance level implies that a new version need to be created if there is a change in the *value* of object's attribute or function. Such attributes can be classified as *version-significant* attributes, which can be updated in only a non-destructive manner by automatically forcing a new version bearing the modified value of that attribute.

Versioning at the template level, on the other hand, means that new versions are created because of some modification in the set of attributes or functions of a particular object, which retains its generic identity in spite of such modifications. Versioning at the template level appears to be a relatively complex task. However, different design databases may require versioning at different levels. The problem is to reconcile these different requirements and to investigate the possibility whether a system can simultaneously support both.

Most computer-aided design applications require the capability to define and store a single object as a collection of other constituent objects; such objects are generally known as *composite objects*. These relationships are hierarchical in nature, where primitive objects form the leaves of the tree.

Time and Versions

In the published literature, versioning and its concomitant version graphs have been discussed without any temporal reference. In this paper, we attempt to explore the issue of temporal significance of version graph, its semantics and ramifications.

Versioned objects can undergo temporal changes in three dimensions: 1. temporal evolution of schemas, where the schema or template of an object changes with respect to time. A related idea is addressed in [Ma87]; 2. changes in some version-significant property that causes new versions; 3. some time-varying attribute [Na88] changes over time. It can be argued that the third dimension of temporal evolution is subsumed by the second if version-significant property or properties were the only time-varying attributes of the object. In this paper, we shall address the temporal evolution in the first two dimensions.

As a general representation, we associate with each version a time-stamp with a begin and end time. The time-interval signifies the period for which the version was a current version. This idea of *currency interval* bears a close analogy with transaction time proposed in literature [Sn85]. A version graph with temporal attributes will be associated with the following: 1. An object, which is the subject of the versions; it will be referred to as a *generic* object. 2. A begin and end time, which gives the currency interval of the version. 3. A set of timed propositions such that each proposition involves the generic object in some term. The effective period of each timed proposition contains the currency interval of the version. 4. A rooted directed acyclic graph where vertices are versions of a common generic object and edges represents successor/predecessor relationships between two vertices with the constraint that the begin time of a successor is never less than that of its predecessor.

The *current version* can be defined as the version whose currency interval includes *Now*. The latest and default versions are selected from the set of current versions. A reference made to a generic object is dereferenced to its default version. This scheme, evidently, allows several versions to be simultaneously current versions; that is, they have overlapping currency periods. A version can be *frozen*. The operation of freezing will involve the following: 1. Assigning the end-

time a constant value, which will be current time in most of the cases. 2. All timed propositions defined on this version get that constant time as their end-time. 3. The frozen version will admit no update. 4. The frozen version will not be allowed to have any new successor. 5. In some applications, it may be necessary at the time of freezing to create a new successor version beginning at the end-time of its frozen predecessor by copying its attributes and time propositions.

The scheme just described appears to have several advantages in the context of design, which is an exploratory and iterative process. It allows several versions to be the current versions; i.e., the versions that are still in the exploratory phase. Versions in this phase can be modified with and without spawning successor versions. Successor and predecessor versions can have overlapping currency periods. A successor, after having been explored, may be found to be unpromising, and thus can be frozen; however, a new version can be created from its predecessor that may still be one of the current versions. Current and frozen versions are defined to be derivable from the currency interval. The successor/predecessor relationship imposes a partial order on the versions of a generic object, whereas association of currency intervals with versions imposes a total order (with respect to its begin time) on them. This fact can be exploited to answer a large number of meaningful queries which can either make reference to explicit time and/or some temporal relationships [Na88] of the versions.

As mentioned earlier, complexity is added in the design database when we consider complex objects and their *current version*. For a complex object to be instantiated for the purposes of simulation or testing, etc., its current version has to be determined. The current version of a complex object is a collection of its primitive constituent objects such that the intersection of the currency periods of their default (or any current) versions includes *Now*. It is quite probable that at some point in the design phase the current version of a complex object is null. These currency intervals can also be used to compute a set of intervals for which the complex object has been current or valid.

Representation Schemes

A simple representation scheme has already been mentioned where versions are represented as a directed rooted acyclic graph. Versions are associated with absolute time intervals represented by vertices of the graph whose edges represent a weak temporal relationship of predecessor/successor.

It has been proposed that sometime it may be worthwhile to represent the temporal relationships between versions in relative terms [Ke87]. In this scheme, vertices of the graph are not associated with absolute time values, but the edges represent different relative temporal relationships -- before, after, overlap, meet, etc -- between the vertices. In the presence of such information, processing of queries may involve finding of transitive closure in addition to arithmetic computations. [Ch88] discusses the computational aspects of such a scheme.

Another interesting idea is of *Reification* [Ge87]. Reification makes it possible to represent relations and functions as objects in the universe of discourse. As a consequence, we can conceptualize attributes of attributes. One can also implicitly represent relationships among temporal relationships, such as subsumption, etc. Extensibility and a simpler representation of quantification are other advantages of this scheme.

We believe that a modified variant of TRM [Nav88] can also be used to represent the temporal and non temporal information in a design database.

Some Open Questions

As long as a versioned object is allowed to have *propositions* involving only its versions and literals, the semantics of this scheme remains well-defined [Ke87]. A propsition is a boolean valued function which can represent both the temporal and non-temporal attributes of and relationships among objects. Representation and semantics tend to become unmanageable when we allow more than one object to participate in a proposition. Consider two versioned objects A and B which participate in a proposition P. For A, P is version significant, whereas for B it is not; that is, for A update has to be done in a non-destructive manner, while only a simple modification of value is needed for B. This may lead to inconsistent representations for B. If B was a non-versioned object, the problem would still remain. The only solution to such a problem is that P has to be a time proposition for B, though B may not require it.

Consider another problem. A versioned product C has a designer D. This fact is represented by a proposition P, which is not version significant. The designer D leaves the company. D cannot be deleted without violating the referential constraint for P. A similar problem arises in object-oriented databases when an object changes its type, where a proposition is defined on the previous type involving some other object. Again, the only feasible solution is to make each proposition a time proposition and process queries with a default time value *Now*, if no explicit time is specified.

Conclusion

In this paper, we explored the temporal aspects of version management. In the literature, versions are generally assumed to have time-stamps; however, their semantics and consequent problems have not been addressed. Several open problems remain in semantic interpretation and representational scheme. It would appear that the temporal dimension is an indispensable part of any database system that deals with complex information that -- directly or indirectly -- involves time. A comprehensive treatment of the issues raised here is the topic of our future work.

References

Ch88 Chaudhuri, S., *Representation of Temporal Relationship*, Technical Report, Stanford University, Stanford, CA (March, 1988).

Co88 Cornelio, A., and Navathe, S. B., *Modeling the Structure and Function of Engineering Design*, Technical Report, University of Florida, Gainesville, FL (August, 1988).

Ge87 Genesereth, M. and Nilsson, N., *Logical Foundations of Artificial Intelligence*, Morgan Kaufman Press (1987).

Ke87 Kent, William, *Versions of Versioning*, Technical Report, Hewlett-Packard Laboratories, Palo Alto, CA (1987).

Ma87 Martin, N., Navathe, S. B., and Ahmed, R., "Dealing with Temporal Schema Anomalies in History Databases", *The 13th Int. Conf. on VLDB*, Brighton, U. K. (1987).

Na88 Navathe, S.B., and Ahmed, R., "A Temporal Relational Model and a Query Language",(to appear) *Information Sciences: An International Journal* (1988).

Sn85 Snodgrass, R., and Ahn, I., "A Taxonomy of Temporal Database", *Proc. ACM SIGMOD Conf.* Austin, Tx (1985).

FUNCTIONALITY OF TEMPORAL DATA MODELS AND PHYSICAL DESIGN IMPLICATIONS

Arie Segev† and Arie Shoshani‡

† School of Business Administration and
Computer Science Research Dept, Lawrence Berkeley Lab
The University of California
Berkeley, California 94720

‡ Computer Science Research Dept.
Lawrence Berkeley Laboratory
University of California
Berkeley, California 94720

Abstract. In previous work, we introduced a data model and a query language for temporal data. The model was designed independently of any existing data model rather than an extension of one. This approach provided an insight into the special requirements for handling temporal data. In this paper, we discuss the main functionality of the model and the implications on physical design. We feel that the functionality and implementation issues discussed in this paper are applicable to all temporal models.

1. INTRODUCTION

In previous papers [Shoshani & Kawagoe 86, Segev & Shoshani 87] we have developed a temporal data model that is independent of existing data models, such as the relational or network models. Our approach differs from many other works [Ariav 86, Clifford & Croker 87, Gadia & Yeung 88, Klopproge 81, Navathe & Ahmed 86, Snodgrass 87, Tansel 86] that extend existing models to support temporal data. Our goal was to design a model (called a *TSC* model for reason that will become clear below) which reflects the semantics and operators of temporal data without being influenced by existing models. Once this model was developed, then it is possible to investigate the incorporation of its structures and operations into existing models. We have performed a requirements analysis for representing the *TSC* model in the context of the relational model [Segev & Shoshani 88]. We showed that the relational model is not sufficient to represent all temporal data and has to be extended with the construct of a temporal relation having new semantic properties. We have also discussed the options for the representation of a temporal relation and the reasons for our preferred representation. In this paper we highlight the functionality of the *TSC* model, and discuss its physical design implications. Note that our reference list is by no means complete; for a complete list, see the bibliography by Stam & Snodgrass in this issue.

2. THE TEMPORAL DATA MODEL

This research was supported by the U.S. Department of Energy Applied Mathematics Sciences Research Program of the Office of Energy Research under contract DE-AC03-76SF00098.

The *TSC* model is based on the simple observation that temporal values are associated with a specific object, and are totally ordered in time; i.e. they form an ordered sequence. For example, the salary history of some individual forms an ordered sequence in the time domain. Accordingly, we introduce the concept of a *time sequence* (*TS*) for a single object (or entity). Further, the set of *TSs* for an object set forms the basic construct of our model which we call a *time sequence collection* (*TSC*). For example, the set of all salary time sequences for a set of employees forms the salary *TSC*. Simple *TSCs* are constructs that have a surrogate, time, and attribute associated with them, denoted as a triplet (S, T, A). For the salary history of employees, this triplet may be (employee_id, month_year, salary). Complex *TSCs* allows multiple attributes to be associated with the same surrogate and time pairs, and are denoted as $(S, T, \bar{A})$. Complex *TSCs* are usually useful for representing synchronous temporal attributes, such as multiple measurements taken at the same time (e.g., air quality measurements: nitrogen, carbon dioxide, etc.)

Naturally, a *TSC* can be accessed on any combination of its surrogate, time, and attribute values. For example, "find the employees that had a salary more than 30k in July 1988" is a query that accesses both the time and attribute components of the salary *TSC*. As will be discussed below, there are some specific assumptions that can be made regarding the access patterns of temporal data which greatly influence the choice of physical structures.

One of the essential conclusions that was reached in our previous work is that *TSCs* should have certain meta-data properties. These properties characterize the semantics and the interpretation of the *TSC* values, and have a profound effect on physical data structures chosen to implement a *TSC*. Below is a brief description of these properties.

2.1. *TSC* properties

There are four properties of interest [Segev & Shoshani 87]:

(1) *Time granularity*: this property specifies the points in time that can potentially have data values. For example, if salaries are assigned at increments of months, then the salary *TSC* granularity is months. Note that time points do not necessarily have values (e.g., salaries do not usually change every month for an individual.)

(2) *Life span*: the life span specifies the range of valid time points of a *TSC*. The life span implies that some physical mechanism is necessary in order to distinguish between existence and non-existence of time points in the *TSC*. The life span definition can be fixed or vary over time. In the latter case, the life span is either extended continuously to "current-time" or it forms a "moving-window".

(3) *Type*: the type of a *TSC* can be thought of as specifying the behavior of time sequences in the *TSC*. For example, a type "step-wise constant" (SWC) can be associated with the salary *TSC*, to mean that the values for time points that do not have explicit data values are determined from the last data value. Other useful types are "discrete" and "continuous", which carry the obvious meaning. A user defined type is also allowed.

(4) *Interpolation rule*: this property is associated with the type. For the SWC type, the rule is as was described above. For the discrete type, the rule is simply that missing values cannot be interpolated. For the continuous type, there is a default continuous function to interpolate values. For the user defined type, an interpolation rule must be supplied by the user.

(5) *Regularity*: A *TSC* is regular if for each time point of the given granularity, a data value has been provided, that is, there is no need to interpolate. This property also provide semantic information to the user; for example, one may want to apply certain statistical analysis only to collected data rather than to interpolated data.

The physical implication of the above properties is that we need to support the interpolation rule according to the *TSC*'s type and granularity. In non-temporal systems the lack of an entry (e.g., there is no entry for some project and some part) implies that the information does not exist. In contrast, in order to support a *TSC* structure, the lack of an entry requires the interpolation of the value. For example, if there is no entry for the salary of John in June 1988, it has to be inferred. As will be discussed later, this requirement affects directly the indexing structures.

2.2. Time points, event points, and change points

Recall that we described time points as points that can potentially have a data value. Time points are defined by the granularity of a *TSC*. As mentioned previously, a time point may have an *explicit* value associated with it, may have an *implicit* value determined by the interpolation rule, or may have no value (*null*). We refer to time points that have explicit data values as *event points*. For example, the time points where a new salary is assigned are considered event points. Clearly, the event points form a subset of the time points. Usually, one would want to store only the event points, and interpolate the values of the other time points when necessary. However, we may wish to interpolate values ahead of time depending on the density of event points, where *event density* is defined as the ratio of event points to time points. In the case of the regular *TSCs* (mentioned above) the event density is 1. As the event density approaches 1, we may prefer to fully populate the *TSC* in order to use simpler data structures and indexing methods. These issues are discussed further in Section 3.

Now, suppose that consecutive event points have the same values. As can often occur with measurement data or statistical data (e.g., consecutive temperature values may be the same). In principle, we could apply a compression operator, so that all repeating "duplicate" values are removed, thus achieving better storage utilization. We call the subset of event points that is left *change points*. One should be careful with compressing out "duplicate values" because it can result in loss of information. For example, two consecutive salary event points may signify "no raise", and the removal of the second event point (which is a duplicate value) would result in the loss of this information. Thus, a compression of duplicates should be a user selected parameter, as it depends on the semantics of the application.

Change points are also important as part of a predicate over time sequences. For example, one may be interested only in the times that employees change departments. In general, all three predicate conditions for finding time, event, and change points should be supported by physical structures.

2.3. Temporal normal forms

In [Segev & Shoshani 88] we discuss the issue of temporal normal forms in the context of a relational representation of the *TSC* model. We argue there that it is appropriate to impose the restriction on *TSCs* that any time slice (at some time point t) would result in a standard First Normal Form relation. We call this condition a First Temporal Normal Form (or 1TNF). The practical implication is that for each instance of a surrogate and time point, each attribute has a single value

(possibly null). While this requirement seems obvious, its enforcement may not be trivial depending on the physical structure chosen for the implementation of a *TSC*. We elaborate on this point in Section 3. A definition of another temporal normal form was proposed by [Navathe & Ahmed 86]. The logical implications of that definition are discussed in [Segev & Shoshani 88].

2.4. Operators

In [Segev & Shoshani 87] we discussed at some length the operators that should be applied to *TSC*s. They include various temporal predicates, aggregate functions in several dimensions, accumulation operators (such as a running average), and operators between multiple *TSCs*. The physical support of such operators is quite complex and is still under study. However, to illustrate the additional complexity that a temporal model introduces we discuss briefly below the implication of the aggregate functions.

Operations over *TSCs* produce a new *TSC*. In particular, operators involving aggregate functions require the determination of a new granularity for the resulting *TSC*, and often a new type as well. For example, we may wish to aggregate over a *TSC* which represents deposits and withdrawals in some bank account, in order to generate a *TSC* for the running balance. The resulting *TSC* will have a type SWC while the original *TSC* is of type discrete. Alternately, getting a monthly sales figure from a daily sales *TSC*, changes from one granularity to another. A more difficult problem arises when we need to decrease the level of time granularity (e.g., estimate daily figures from monthly figures, assuming we know daily patterns). This problem is referred to as the "disaggregation problem" by statisticians. Our work on these problems is in progress, and its description is beyond the scope of this paper.

3. PHYSICAL DESIGN IMPLICATIONS

The problem of physical design of temporal databases still deserves a significant attention. In this section we discuss physical design issues, some of which has been addressed by works such as [Ahn & Snodgrass 86, Gunadhi & Segev 88a, Lum et al 84, Rotem & Segev 87]. At an abstract level, physical design of temporal databases is similar to conventional database design in the sense that the desirable structure is a function of application requirements. There are, however, several important differences. For example, in temporal databases, one expects that many queries will need the data ordered by the time attribute; also, the data itself is likely to be either static or append-only. We will first classify elements of the environment that affect the design, and list the major physical design choices. We will then discuss some of the combinations of environmental parameters and physical design choices.

3.1. Environmental parameters

The following is a list of the parameters that affect the physical design.

(1) *Static* vs. *dynamic* data. Static data represents non-updatable historical data with a fixed life span, while dynamic data arise in cases where temporal data have a variable life span or a fixed life span with updatable data (i.e., missing or incorrect data). In the case of dynamic historical data, an important aspect is its append-only nature.

(2) *Sparse* vs. *dense* data. We define two measures of density that have physical design implications. The first measure, *event density* (introduced in Section 2), is the ratio of the number of event points to the number of time points. The second measure is *existence density*,

defined as the ratio of the number of existence points to the number of time points, where an existence point is a time point that has a non-null attribute value (either explicit or implicit). For a discrete sequence, all existence points are event points, and thus the two density measures are the same. However, for a *SWC* or a general continuous sequence, the two measures will differ if the two sequences contain null values.

(3) *Integration of temporal and non-temporal data.* In the case where such an integration is required, a priority may be given to the processing of non-temporal data and the current portion of the temporal data.

(4) *Query types.* This is the most influential parameter as far as the physical design is concerned. Queries can be classified as follows: i) Single-*TSC* queries; either point or range queries. The qualification part can refer to S, T, or A, or any combination. Note the T qualification can be either a value (e.g, June 3, 1988) or a relative position (e.g., third from the beginning). ii) Multi-*TSC* queries. These queries involve some kind of a join plus any of the qualifications of single-*TSC* queries.

3.2. Physical design options

The physical design options can be classified as follows.

(1) *Record structure.* We consider here the possible options for extending existing relational systems. In order to minimize changes to current systems, we have chosen to have a normalized tuple as the physical data unit.

(2) *Sorting.* A major issue is whether to have the data sorted by time (either physically or logically), and if so, whether the time will be the major sort key or a minor (where the primary order is by surrogate).

(3) *Indexing structures.*

i) *Single attribute indexing.* The two main choices here are hashing (which also determines the placement of data records) or trees (e.g., B-trees).

ii) *Multi-attribute indexing.* There are several options for providing indexing structures:

a) *Separate structures.* In this case each attribute is indexed separately. A multi-attribute access can be done by intersecting pointers from multiple indices.

b) *Joint structures.* The indexing is done directly on the combination of several attributes.

c) *Multi-level structures.* Separate structures are combined in a hierarchical fashion. For example, the leaves of a B+ tree index of one attribute point to a B+ tree index of another attribute.

The above structures have one or more component. In general, each component can be one of two indexing types: either a tree (e.g., R-trees) or a non-tree (e.g., grid files).

The foregoing taxonomy of environmental parameters and physical structures, though general, serves as a helpful framework for investigating the physical design problem. In this paper, we discuss only some the issues; the complete discussion is given in [Gunadhi & Segev 88a and 88b]. It should be noted that we are interested in databases containing a large number of history records. Below is a summary of our conclusions.

3.3. Query types

We anticipate that most of the temporal queries will qualify on S and T (i.e., data records for surrogates at specific time points); if T is absent from the qualification, it means that the query needs the whole history. Moreover, we don't anticipate frequent range qualifications on S. A qualification on A is likely to be combined with either a T or ST qualification (e.g., the data of employees who earned more than 30K at a certain time point). Consequently, range queries are more likely to qualify on T and/or A. In databases which are event-oriented, queries that qualify only on T (i.e., what happened at certain time points?) are more likely; in our analysis, however, we assume that these queries are secondary in importance.

The above discussion implies that we are interested in supporting primarily queries that qualify on ST (note that this is the primary key of the data tuples), or TA, or STA.

3.4. Ordering of the data records

Given the above query types we exclude a primary order of data records by time, but rather have the data sequenced primarily by surrogate and secondarily by time. If the data is static, physical clustering is the best choice; if it is dynamic, an indexing structure may be required to enable access to data in the desirable order.

Note that if the query type do not qualify on the surrogate (i.e., T or TA qualifications), then a primary order by time is desirable. Furthermore, in this case, since the data is append-only we can keep it in contiguous blocks (a clustered index) with a fill factor of (or approximately of) 100%.

3.5. A static environment

If the data is static, dense, and without many contiguous duplicate values, then the optimal solution is to have it organized as a two dimensional array linearized row-wise (rows and columns represent S and T respectively). An index on A may still be required. If the data is sparse, a direct access by ST cannot be calculated, and an index is required (we exclude hashing on ST because of the high likelihood of range qualifications on T). In the case where there are many contiguous duplicate values, compression is useful. It should be noted, however, that the compression scheme should preserve the distinction between event points and interpolated points. The only exception to this is a regular TSC where we know that all points are event points.

3.6. A dynamic environment

Given that we are not interested in T as the primary ordering attribute, physical clustering in order of ST may cause a problem. If the database is append-only and the rate of growth is low, an indexed-sequential organization is appropriate. If the rate of growth is high, long overflow chains my result, and an unclustered index should be used.

3.7. Indexing

The most important environmental parameter that affects indexing is the query types. Our current conclusions (the details are given in [Gunadhi & Segev 88a and 88b]) are the following:

(1) For a large number of surrogates and a significant number of queries that qualify on a single S, we excluded a grid type file for ST or STA dimensions. The reason is that there will be a partitioning line for each surrogate value, and if the partitioning information fits in main

memory, it implies that we can also store the multi-level *S/T* index trees in main memory. A grid type file can be constructed on the *ST* or *STA* dimensions if there are range queries on *S* and queries that qualify on a single *S* can be compromised. Grid file types are a viable option on the *TA* dimensions (e.g., the brick file in [Rotem & Segev 87]).

(2) For *ST* qualifications we prefer to have a multi-level B+ tree index, where the leaves of the *S* tree point to *T* trees. This choice enables integration of historical data with current and non-temporal data such that updates to the latter are separable from the temporal structures. In addition, the total storage cost is frequently lower for the multi-level structure.

(3) It is inefficient to have time index on points other than event points (in the case of a *SWC* sequence, indexing event points is equivalent to indexing time intervals, where the terminal points of an interval are event points). The reason is that we can interpolate in order to find values at points other than event points.

(4) For a *TSC* with a high existence density, there is no point in having a separate *T* index (in the case of a multi-level index a *T* index under an *A* or *S* index can still be useful). The reason for the above observation is that in a *T* index with high existence density, a *T* value of the index will point to data records in a non-selective way. The extreme case is when the the density is 1, and thus for each time point in the lifespan all the surrogates have a value.

3.8. 1TNF Enforcement

As was discussed in Section 2, enforcing 1TNF of a *TSC* implies that only one attribute value is allowed for a given combination of *S* and *T* values.† The mechanism for enforcing 1TNF is dependent on the record structure. For example, there are proposals that claim that it is more desirable for access efficiency to store time sequences as tuples containing intervals. If such a structure is chosen, then a mechanism has to be added to ensure that intervals of the same surrogate do not overlap, so that the 1TNF condition is not violated. On the other hand, if one choses to store tuples with time points rather than intervals, then this condition can be simply enforced by existing mechanisms of the relational model, i.e. by defining the surrogate and the time as a composite key.

REFERENCES

[Ahn & Snodgrass 86]
Ahn I., Snodgrass R., Performance Evaluation of a Temporal Database Management System, *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 1986, pp. 96-107.

[Ariav 86]
Ariav, G., A Temporally Oriented Data Model, *ACM Transactions on Database Systems*, 11, 4(Dec. 1986), pp. 499-527.

[Clifford & Croker 87]
Clifford, J., Croker, A., The Historical Relational Data Model (HRDM) and Algebra Based on Life Spans, *Proceedings of the Third International Conference on Data*

† In the case of $\overline{A}$, only a single combination of $\overline{A}$ values is allowed for a given combination of *S* and *T* values.

Engineering, pp. 528-537. February, 1987.

[Gadia & Yeung 88]

Gadia, S.K., Yeung C-S., A Generalized Model for A Relational Temporal Database, *Proceedings of the ACM SIGMOD International Conference on Management of Data*, June 1988, pp. 251-259. 1986, pp. 390-397.

[Gunadhi & Segev 88a]

Gunadhi H., Segev A., Physical Design of Temporal Databases, *Lawrence Berkeley Lab* Technical Report LBL-24578, 1988.

[Gunadhi & Segev 88b]

Gunadhi H., Segev A., Indexing Structures for Temporal Databases, *Lawrence Berkeley Lab* Technical Report. In progress.

[Lum et al 84]

Lum, V., Dadam, P., Erbe, R., Guenauer, J., Pistor, P., Walch, G., Werner, H., Woodfill, J., Designing DBMS Support for the Temporal Dimension, *Proceedings of the ACM SIGMOD International Conference on Management of Data*, June 1984, pp. 115-130. March, 1986.

[Navathe & Ahmed 86]

Navathe, S.B., Ahmed, R., A Temporal Relational Model and a Query Language, *UF-CIS Tech. Report TR-85-16, Univ of Florida*, April 1986.

[Rotem & Segev 87]

Rotem, D., Segev, A., Physical Organization of Temporal Data, *Proceedings of the Third International Conference on Data Engineering*, pp. 547-553. February, 1987.

[Segev & Shoshani 87]

Segev, A., Shoshani, A., Logical Modeling of Temporal Databases, *Proceedings of the ACM SIGMOD International Conference on Management of Data*, May 1987, pp. 454-466.

[Segev & Shoshani 88]

Segev, A., Shoshani, A., The Representation of a Temporal Model in the Relational Environment, an invited paper to the *4th International Conference on Statistical and Scientific Database Management*, Rome, June 1988.

[Shoshani & Kawagoe 86]

Shoshani, A., Kawagoe, K., Temporal Data Management, *Proceedings of the International Conference on Very Large Databases*, August 1986, pp. 79-88.

[Snodgrass 87]

Snodgrass, R., The Temporal Query Language TQuel *ACM Transactions on Database Systems*, 12, 2, June 1987, pp. 247-298.

[Tansel 86]

Tansel, A.U., Adding Time Dimension to Relational Model and Extending Relational Algebra, *Information Systems*, 11, 4 (1986), pp. 343-355. *ACM Transactions on Database Systems*, 12, 2, June 1987, pp. 247-298.

NON FIRST NORMAL FORM TEMPORAL RELATIONAL MODEL

Abdullah U. Tansel

Bernard M. Baruch College
17 Lexington Avenue, Box 513
New York, N.Y. 10010

1. Introduction

In most databases, when an object's attributes assume new values, their previous values are discarded from the database. Such databases carry only current data. However, in many real life applications, there is a need for both current and historical data. For instance, in a corporate database environment, the database should possess a temporal dimension for supporting managerial information needs.

It is possible to consider time as a special attribute in different data models and to use its values as the time reference of other attributes. However, this is a rather ad hoc and limited solution. It either creates undue data redundancy and/or provides limited time processing capacity. It seems to us that the solution to this problem lies with either developing a new model to support the time dimension or augmenting existing data models to support the time dimension in a coherent way. In this paper we adopt the second approach and propose an extension to nested (Non First Normal Form - N1NF) relations for handling time variation of complex objects.

Recently, considerable research effort has been directed to temporal aspects of information systems and research in this area has proliferated. A comprehensive summary [Snodgrass 86] and bibliography [McKenzie 86] appeared in a recent issue of SIGMOD Record. Many proposals to extend relational model [Codd 70] for time support have been made. One group of researchers adds special time attributes to flat (1NF) relations [Ariav 86, Ben-Zvi 82, Clifford & Warren 83, Jones 84, Lum et al 84, Navathe & Ahmed 87, Snodgrass 87]. Another group proposes using N1NF relations and time-stamping attributes in contrast to the time-stamping of tuples found in the former approach [Gadia 87, Tansel & Clifford 85, Tansel 86]. Gadia extracts snapshots from historical relations whereas Tansel either applies algebra operations directly or normalizes (flattens) the relations before extracting data. In both approaches, there is only one level of nesting in the model. Later, Gadia relaxed the homogeneity restriction and generalized his approach [Gadia 88]. In a recent proposal, McKenzie and Snodgrass added time stamps to the attributes of flat relations [McKenzie & Snodgrass 1987]. The time stamps are sets of time points. Lorentzos and Johnson proposed an extension which adds two time points for any set of attribute [Lorentzos & Johnson 87]. Other research in temporal databases includes [Ginsberg 84, Klopprogge & Locdkemann 83, Anderson 81, Shoshani & Kawagoe 86].

Nested (N1NF) relations organize data hierarchically. Each tuple component can be an atomic (simple) value or another relation. Two new operations **unnest and nest** have been introduced to relational algebra for restructuring nested relations [Jaeschke & Schek 82]. Pack and unpack are similar to one attribute nest and unnest operations used on set valued relations [Ozsoyoglu, Ozsoyoglu, and Matos 87].

In this paper, we combine the research in temporal databases and nested (N1NF) relations for nontraditional database applications like CAD/CAM, office automation, etc. We model a simple attribute value as a temporal atom which consists of two components, a temporal set and a value. The temporal set denotes a time period and the temporal atom asserts the value was valid over this period. We also discuss redundancy in nested temporal relations and develop criteria for well-structured nested temporal relations. Our previous work modelled entity histories. This extension to relational model allows to model histories of relationships as well.

Section 2 describes the model and the scheme trees associated with nested temporal relations. Section 3 covers redundancy in nested temporal relations and establishes criteria for structuring such relations. Section 4 is the conclusion.

2. N1NF Temporal Relations

2.1. Preliminaries

Let T be set of time points mapped to natural numbers, i.e. {0,1,2,...,n} which is well ordered $\leq$ relationship. 0 is the relative beginning point. The symbol n denotes the present time instant and its value increments as the clock ticks. We do not consider a time unit. It is user defined and can be any of seconds, minutes, hours, days, etc. A time interval is a set of consecutive time points. For instance, [l,u) is a time

interval including the time points between 1 and u, and 1 but not u. A temporal set (called a temporal element in [Gadia & Vaishnav 85]) is a set of time points which can be grouped into disjoint time intervals. Two example of temporal sets are {[5,10)} and {[5,10), [20,30)}.

2.2. The Model

The fundamental construct of our model is a temporal atom, <t,v> where t is a temporal set and v is a value. The temporal atom asserts that the value v is valid over the time points represented by the temporal set t. The temporal set can be considered as a union of disjoint time intervals [Gadia 85, Gadia 87, McKenzie and Snodgrass 87] or as a single interval [Tansel 85, Tansel 86]. In the latter case several temporal atoms are used in place of the one temporal atom of the former case. For instance, the temporal atom <{[5,10), [20,30)}, Tom> is represented as two separate temporal atoms: <{[5,10), Tom> and <{[20,30), Tom>. These two representations are equivalent in terms of their information content, however, the first representation is more concise. Algebraic operations are defined accordingly depending on the representation method chosen [Garnett & Tansel 88, Tansel & Garnett 88].

A temporal atom represents one value in the history of an attribute. Thus, the history of an attribute is represented as a set of temporal atoms. Furthermore, attribute values may also be relations whose tuple components are made up of temporal atoms or previously defined relations. Hence, we relax the requirement of first normal form (flat) relations. The resulting temporal relation is N1NF and we call it a nested temporal relation (NTR). In such a relation there may be time invariant attributes, e.g. the social security number. These attributes can be represented either as a single value or as a temporal atom where the temporal set is the existence period of the concerned object. For the Attribute A, A_T and A_V represent its temporal set and value components, respectively.

Figure 1 gives an example nested temporal relation, COURSES. The attributes of COURSES are course number (Cno), course names (CnameX), sections of the course (SectionX) and the text books used (TextX). Attributes whose values are relations have a name ending with the letter "X". Cno is a simple attribute whose values are temporal atoms. The remaining attributes are relations. CnameX and TextX are unary relations. SectionX, on the other hand, is another temporal relations whose attributes are section identifier (Section), students enrolled in that section (StudentX) and teachers of a section (TeacherX). The remaining attributes are self explanatory. COURSES relation has two tuples, the first one for the course C1 and the second one for course C2. The unary relation CnameX has two 1-tuples (temporal atoms). SectionX consists of two 3-tuples the first of which has two sections, and so on.

2.3 Scheme Trees

The attributes of a nested temporal relation are organized as a "scheme tree". The name of the nested temporal relation is the root of the scheme tree. Each descendent of the root is either a simple attribute name or another tree which is also a nested temporal relation. The leaves of the scheme tree are simple attributes and non-terminal nodes are called high-order names. High-order names represent temporal relations. A high order name may consist of simple attributes and/or other high order names. The high order names are recursively defined but do not contain cycles (loops), i.e., they are strictly hierarchical. Figure 2 gives the scheme tree of the nested temporal relation depicted in Figure 1.

The leaves of the scheme tree carry the time-stamps for the time-varying attributes. The non-terminal nodes do not carry any time reference. The time reference of a non-terminal node (a higher order name) can be induced recursively from the time reference of its descendants. Let X be a non-terminal node and Y1, Y2,..., Yn be its descendants. Then, the time reference of X is $X_T = Y1_T \cup Y2_T \cup ... \cup Yn_T$. The time reference of each Yi is similarly defined until leaf nodes are reached. For instance $studentX_T = Sno_T \cup Sname_T \cup GradeX_T$ where $GradeX_T = Grade_T$. Similarly, time reference of SectionX is $Section_T \cup StudentX_T \cup TeacherX_T$. Thus, time reference of each high order attribute name is determined by the time reference of its descendants. This is an integrity constraint for the nested temporal relations.

Each simple attribute A of a nested temporal relation has an associated domain of values denoted as DOM (A). DOM (A) contains either temporal atoms or atomic values. For the sake of uniformity in the remainder of the paper, we will assume that attributes are defined on domains consisting of temporal atoms only.

COURSES

Cno	CnameX		SectionX						TextX
	Cname	Section	StudentX			TeacherX			Text
			Sno	Sname	GradeX	Tno	Tname	SalaryX	
					Grade			Salary	
<0,n,C1>	<0,5,DS> <5,n,Int to DS>	<0,2,Sel>	<0,1,S1>	<0,1,Tom>	<0,1,A>	<0,5,T1>	<0,5,Tom>	<0,2,20K> <2,5,25K>	<0,n,Data Structures>
			<0,1,S2>	<0,1,Ann>	<0,1,B>				
			<0,2,S3>	<0,2,Bob>	<0,1,F> <1,2,A>				
			<1,2,S4>	<1,2,Tim>	<1,2,C>	<5,n,T2>	<5,n,Liz>	<5,6,20K> <6,8,25K> <8,n,30K>	<0,n,Algorithms>
			<0,1,S5>	<0,1,Ann>	<0,1,A>	<0,n,T3>	<0,n,Hall>	<0,6,20K> <6,n,30K>	
			<0,1,S6>	<0,1,Gary>	<0,1,B>				
			<5,6,S7>	<5,6,Dan>	<5,6,B>				
		<0,n,Se2>	<8,9,S8>	<8,9,Bill>	<8,9,A>				
<0,6,C2>	<0,6,Int to DS>	<0,6,Sel>	<5,6,S1>	<5,6,Tom>	<5,6,A>	<0,6,T2>	<0,6,Liz>	<0,5,15K> <5,6,25K>	

Figure 1. An Example Nested Temporal Relation.

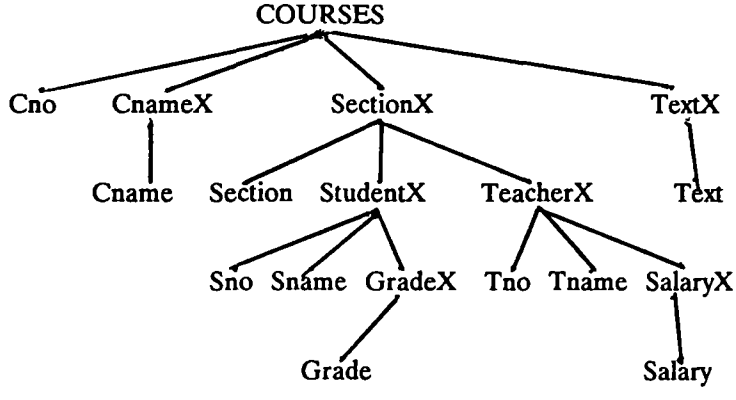


FIGURE 2. Scheme Tree for COURSES Relation

3. Structuring Nested Historical Relations

Nested temporal relations can be structured in various ways. Each different structure has a different level of data redundancy. Of course, controlled and reduced redundancy is desirable to avoid anomalies in the insertion, deletion and update operations. This is the motivation behind normalization in relational database design theory based on functional and multivalued dependencies. A similar normalization theory has been developed for nested relations in [Ozsoyoglu & Yuan 87, Roth & Korth 87]. In what follows, we adopt this approach for structuring nested temporal relations.

From a temporal perspective, attributes of a relation can be classified into different categories with respect to the type and number of values they assume [Shoshani & Kawagoe 86, Segev & Shoshani 87]. An attribute which takes a single value at any time is called stepwise constant. For instance, the salary of an employee is an example for this type of attributes. There is only one salary value at any time. When a new salary value is assumed as the current value, the previous value becomes no longer valid. Such an attribute in traditional relational theory represents a functional dependency on the relation's key. However, when temporal dimension of the database is considered it becomes a multivalued dependency on the relation key. There is a set of attribute values (one is valid for a period of time) for each key value (multivalued dependency). Discrete attributes assume a single value at a time point which is valid only at that point but not any other time. In this case, the temporal set consists of a single time point. Values of such attributes also represent functional dependencies at one time point but multivalued dependencies over a time period.

Another group of attributes, (also termed stepwise constant or discrete), can take a set of values at any time instant. Skills of an employee, is an example for this attribute type. An employee has zero one or more skills at any time. This represents a multivalued dependency. It is still a multivalued dependency when viewed in a time perspective.

Let's consider the functional and multivalued dependencies for the COURSES relation. These dependencies are given below in Figure 4. Note that Figure 4.a lists the dependencies in the static case without any time dimension. Figure 4.b gives the same dependencies in a temporal perspective. That is, the attribute values are temporal atoms. Cno is time invariant; there is only one Cno value throughout the database history for a course. Although we represent it as a temporal atom, it can be considered as a simple (single) value for all purposes. On the other hand, there is a well-defined set of temporal atoms for Cname attribute, So, the functional dependency, Cno --> Cname of the static case turn into a multivalued dependency, Cno -->> Cname in the temporal database. Other dependencies are similarly explained.

```
Cno -->  Cname
Cno -->> Text
Cno -->> Section,Sno,Sname,Grade,
         Tno,Tname,Salary
Cno,Section -->> Sno,Sname,Grade
Cno,Section -->> Tno,Tname,Salary
Cno,Section,Sno --> Grade
Sno --> Sname
Tno --> Tname
Tno --> Salary
```

a. without time

```
Cno -->> Cname
Cno -->> TExt
Cno -->> Section,Sno,Sname,
         Grade,Tno,Tname,Salary
Cno,Section-->>Sno,Sname,Grade
Cno,Section->>Tno,Tname,Salary
Cno,Section,Sno-->> Grade
Sno--> Sname
Tno--> Tname
Tno-->> Salary
```

b. with time

Figure 4. Functional and Multivalued dependencies in the COURSES Relation.

A different definition for the scheme trees is given in [Ozsoyoglu & Yuan 87]. This definition constructs the scheme tree for a nested relation with respect to functional and multivalued dependencies among the attributes. Attributes are nodes of the scheme tree, dependencies among the attributes are its edges. The ancestors of a node multidetermine (functionally determine) its descendants. The scheme tree for the COURSES relation is given in Figure 5. Note that it does not contain high order names, but is equivalent to the scheme tree given in Figure 2 in terms of its information content. They represent the same nested temporal relation structure.

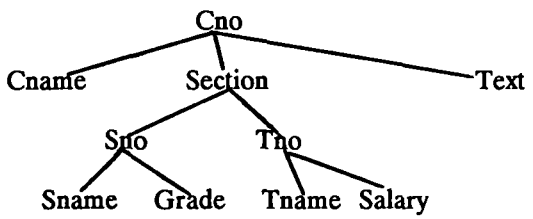


Figure 5. Scheme Tree for Courses Relation

The COURSES relation contains redundancies. For instance, salary data of a teacher is repeated if a teacher taught several different sections. Two tuples are created for the teacher T2 and relevant parts of salary are split in these tuples. T2 taught both section Se1 and Se3 during [5,6) and his salary for this period is repeated. Furthermore, salary history of T2 has been split into two tuples. Including the entire salary history of T2 in each tuple would still create problems.

Another source of data redundancy is the repetition of Tname or Sname. If a teacher taught several sections, possibly at different times teacher's name would be repeated in different tuples. This redundancy is due to the structure of the scheme tree which does not represent the dependencies among the attributes properly. The first type of redundancy is caused by the partial dependency of Salary on Tno. The scheme tree implies that (Cno, Section, Tno) multidetermines Salary which can be obtained from Tno -->> Salary by augmentation. The other data redundancy is also caused by a partial functional dependency. That is, (Cno, Section, Tno) --> Tname can be obtained from Tno --> Tname by augmentation.

A scheme tree is called a normal scheme tree if it does not contain any partial and transitive dependencies [Ozsoyoglu & Yuan 87]. These anomalies are eliminated by decomposing COURSES relation to remove undesirable dependencies and creating a scheme forest. Figure 6 depicts the resulting scheme forest for COURSES relation. The nested historical relation schemes in the scheme forest are given in Figure 7.

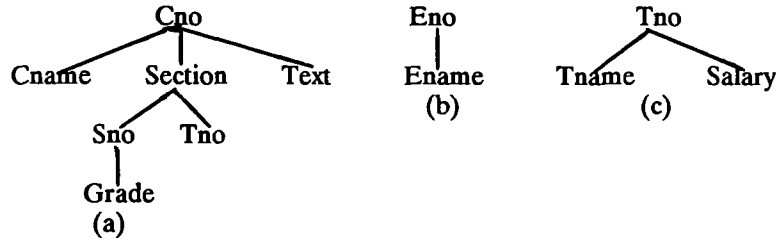


Figure 6. Scheme Forest for COURSES Relation

A root to leaf path in a normal scheme tree represents a 4NF decomposition. The scheme forest of Figure 6 implies 6 relations which are all in 4NF. These are (Cno, Cname), (Cno, Section, Sno, Grade), (Cno, Section, Tno), (Cno, Text), (Eno, Ename), and (Tno, Tname, Salary). Note that (Tno, Tname) is not represented as a separate relation since Tname is functionally determined by Tno. Attributes of these 4NF relations are temporal atoms. The time reference can be factored out from the attributes and can be added to the tuples. This is the approach followed by various researchers as indicated in Introduction.

C1

Cno	CnameX	SectionX		TextX
	Cname	StudentX	TeacherX	Text
		Sno GradeX	Tno	
		Grade		

C2

Eno	Ename

C3

Tno	Tname	SalaryX
		Salary

Figure 7. Nested Relation Schemes in the Scheme Forest

5. Conclusion

There are a few points we wish to touch on briefly. Nested temporal relations may appear to be complicated at first glance. However, when they are properly structured in the form of normal scheme trees to avoid data redundancy, the relations exhibit a natural and intuitive conceptualization of the data. The nested temporal relation represents the associations among entities, perhaps at several levels, depending on the type of associations. Attributes of an entity will be represented at only one level of nesting. This representation is also equivalent to a 4NF decomposition, as is shown in [Ozsoyoglu & Yuan 86]. The same is also true for the nested temporal relations. Instead of creating many small 4NF relations, the data is represented in a concise manner as a scheme forest.

One criticism is that a series of unnest operations is needed to extract information from nested temporal

relations. This is not always the case, because local versions of the operations can be defined. They allow extraction of data without any unnest operations for some queries. Furthermore, it is possible to optimize the query processing methodology and to process nest/unnest operations together with other algebra operations, e.g., selection or join operations.

The user interacts with the database by a user-friendly query language. For instance, versions of SQL for nested relations have already been proposed [Dadam et al., 85]. Similar enhancements can be added to SQL for nested historical relations as well. Moreover, a graphical query language can also be modified for this purpose i.e., Time-by-Example [Tansel, Arkun & Ozsoyoglu 87]. All these points are currently under investigation and we expect to report the results in future papers.

Please note that the nested historical relations can be structured in different ways if the associations among the entities are many to many. Different structures would be more convenient for different groups of users. When flat relations are used to represent association, the entities are equally represented. On the other hand, a nested historical relation arranges the entities in a hierarchy and groups the data according to this hierarchical structure. One form of nested historical relation can be obtained from another by a sequence of unnest/nest operations.

References

[Anderson 81] Anderson, T.L., 'The Database Semantics of Time', Ph.D. Diss. Univ, of Washington,1981.

[Ariav 86] Ariav, G., 'A Temporally Oriented Data Model', ACM TODS, Vol 11, No.4, 1986, pp: 499-527.

[Ben-Zvi 82] Ben-Zvi, J., 'The Time Relational Model',Ph.D. Diss., UCLA, 1982.

[Clifford & Warren 83] Clifford, J., Warren, D.S., 'Formal Semantics for Time in Databases', TODS, Vol. 6, No.2, June 1983, pp: 214-264.

[Codd 70] Codd, E.F. 'A Relational Model of Data for Large Shared Databanks', Comm, of the ACM, Vol. 13, No. 6, June 1970, pp: 377-387.

[Dadam, et al., 85] Dadam, P.M., et al., 'A DBMS Prototype to Support NF2 Relations: An Integrated View on Flat Tables and Hierarchies', ACM SIGMOD Conf., 1986, pp: 356-367.

[Gadia & Waishnav 85] Gadia S.K., Yeung, C.S. 'A Query Language for Homogeneous Temporal Databases', Proc. ACM PODS Conf., 1985, pp: 51-56.

[Gadia 86] Gadia, S.K. 'A Multi Homogeneous Model for Temporal Databases', 2nd Int. Conf. on Data Engineering, 1986.

[Gadia 87] Gadia, S.K., 'A Homogeneous Model and Query Languages for Temporal Databases'. To appear in TODS, 1987.

[Gadia & Yeung 88] Gadia, S.K., Yeung, C.S., 'A Generalized Model for a Relational Temporal Database', Proc. ACM SIGMOD Conf., 1988.

[Garnett & Tansel 88] Garnett, L., Tansel, A.U., 'Equivalence of the Relational Algebra and Calculus Languages for Nested Relations', Technical Report, Baruch College CUNY, 1988. (in preparation).

[Ginsberg & Tanaka 84] Ginsberg, S., Tanaka, K., 'Computational Tuple Sequences and Object Histories', Proc. of VLDB, 1984, pp: 208-217.

[Jaeschke & Schek 82] Jaeschke, G., H. Schek, 'Remarks on the Algebra of non First Normal Form Relations', Proc. of ACM PODS Conference, 1982.

[Lorentzos & Johnson 87] Lorentzos, N.A., Johnson, R.G., 'A Model for a Temporal Relational Algebra', Proc. of TAIS Conf., 1987, pp: 99-113.

[Jaeschke 84] Jaeschke, G., 'Nonrecursive Algebra for Relations with Relation Valued Attributes', Proc. of ACM PODS Conference, 1982, pp: 228-239.

[Lum, et al., 84] Lum, V., et al., 'Designing DBMS Support for the Temporal Dimension', Proc. of ACM SIGMOD Conference, 1984.

[McKenzie 86] McKenzie, E., 'Bibliography: Temporal Databases', ACM SIGMOD Record, Vol. 15, No. 4, pp: 40-52.

[McKenzie & Snodgrass 87] McKenzie, E., Snodgrass, R. 'Supporting Valid Time: an Historical Algebra', Technical Report; Computer Science Department, Univ. of North Carolina, at Chapel Hill, 1987.

[Navathe & Ahmed 87] Navathe, S.B., Ahmed, R., 'TSQL - A Language Interface for History Databases', Conference on Temporal Aspects of Information Systems', 1987, pp: 113-128.

[Ozsoyoglu, Ozsoyoglu, & Matos 87] Ozsoyoglu, G., Ozsoyoglu, M.Z., Matos, V., 'Extending Relational Algebra and Relational Calculus with Set-valued Attributes and Aggregate Functions', TODS, Vol. 12, No. 4, 1987, pp: 566-597.

[Ozsoyoglu & Yuan 87] Ozsoyoglu, M.Z., Yuan, Li-Yan, 'A New Normal Form For Nested Relations', ACM TODS, Vol. 12, No. 1, 1987, pp: 111-136.

[Roth & Korth 87] Roth, M.A., Korth, H.T., 'The Design of -1NF Relational Databases into Nested Normal Form', Proc. of ACM SIGMOD Conf., 1987, pp: 143-154.

[Segev & Shoshani 87] Segev, A., Shoshani, A., 'Modelling Temporal Semantics', Temporal Aspects of Information Systems Conf., 1987.

[Shoshani & Kawagoe 86] Shoshani, a., Kawagoe, K., 'Temporal Data Management', VLDB 12, Kyoto, Japan, 1986.

[Snodgrass 86] Snodgrass, R., 'Research Concerning Time in Databases: Project Summaries', ACM SIGMOD Record, Vol. 15, 1986, pp: 19-39.

[Snodgrass 87] Snodgrass, R., 'The Temporal Query Language, TQL', TODS, Vol. 12, No. 2, 1987, pp: 247-298.

[Tansel & Clifford 85] Tansel, A.U., Clifford, J., 'On a Historical Relational Algebra: Two Views', Proc, of the ACM SIGMOD Conference, 1985, pp: 247-264.

[Tansel 86] Tansel, A.U., 'Adding Time Dimension to Relational Model and Extending Relational Algebra', Information Systems. Vol 13, No. 4, 1986, pp: 343-355.

[Tansel, Arkun, & Ozsoyoglu 87] Tansel, A.U., Arkun, M.E., Ozsoyoglu, G., 'Time-by-Example Query Language for Historical Databases', to appear in IEEE Transactions on Software Engineering, 1987.

[Tansel 87a] Tansel, A.U., 'A Statistical Inference to Historical Relational Databases', Proc. of 3rd International Conference on Data Engineering, 1987.

[Tansel & Garnett 88] Tansel, A.U., Garnett, L., 'Relational Algebra of Nested Historical Relations', manuscript in preparation, 1988.

A Bibliography on Temporal Databases

Robert B. Stam and Richard Snodgrass

Department of Computer Science
University of North Carolina
Chapel Hill, NC 27514

1 Introduction

This bibliography is an update of a 1986 bibliography (McKenzie, E. *Bibliography: Temporal Databases, ACM SIGMOD Record*, 15, No. 4, Dec. 1986, pp. 40-52), which was in turn an update of a 1982 survey (Bolour, A., T.L. Anderson, and H.K.T. Wong, *The Role of Time in Information Processing: A Survey, SIGArt Newsletter*, 80, April 1982, pp. 28-48) This bibliography consists of papers published or accepted for publication since the previous bibliography, as well as older papers that have not appeared in these previous surveys.

The pre-1982 survey, covering 1960-1982, contained 16 papers specifically relating time to database management, with 5 appearing in journals. The McKenzie bibliography, covering the next five years (1982-1986), listed over 80 papers on this subject, with 10 appearing in journals. This bibliography, covering the next 21 months (Jan. 1987 through September, 1988), lists over 100 papers, with 18 appearing in journals. During this period, the first conference on the topic was held (TAIS: Conference on Temporal Aspects in Information Systems, May, 1987). Our conclusion is that the field is growing rapidly and seems to be maturing, as more papers appear in journals (Figure 1 illustrates that the growth remains exponential).

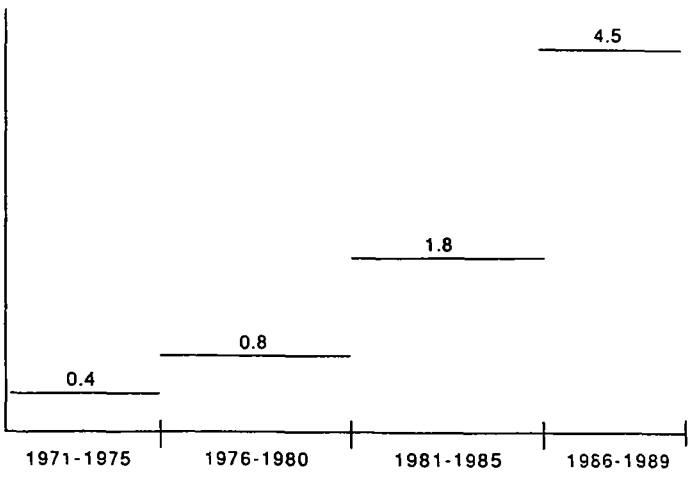


Figure 1: Average Number of Journal Papers Appearing Annually

The entries in this bibliography are classified according to a taxonomy of time in databases developed by Snodgrass and Ahn (*Temporal Databases, IEEE Computer*, 19, No. 9, September, 1986, pp. 35-42.) Papers that propose augmenting conventional database management systems to represent *transaction time* (that is, the time when information is entered into a database) are listed in Section 2. Papers that propose augmenting conventional database management systems to represent *valid time* (that is, the time that information models in the real world) are listed in Section 3. Finally, papers that propose augmenting conventional databases with both aspects of time are listed in Section 4.

We apologize for omission or misclassification of papers. We thank Ed McKenzie for help in locating the papers listed here. This research is supported in part by the Office of Naval Research, Contract N00014-86-K-0680, and by the National Science Foundation under Grant DCR-8402339.

2 Transaction Time

As defined above, transaction time concerns the time when data is entered into the database. Relations that contain transaction time are termed *rollback* relations.

Abiteboul, S. and V. Vianu. *Transactions and Integrity Constraints*, in *Proceedings of the ACM Symposium on Principles of Database Systems.* 1985, pp. 193-204.

Abiteboul, S. and V. Vianu. *Deciding Properties of Transactional Schemas*, in *Proceedings of the ACM Symposium on Principles of Database Systems.* 1986, pp. 235-239.

Abiteboul, S. and V. Vianu. *A Transaction Language Complete for Database Update and Specification*, in *Proceedings of the ACM Symposium on Principles of Database Systems.* San Diego, CA: Mar. 1987.

Adiba, M.E. and N. Bui Quang. *Dynamic Database Snapshots, Albums, and Movies*, in *Proceedings of the Conference on Temporal Aspects in Information Systems.* AFCET. France: May 1987, pp. 207-225.

Barbic, F. and R. Maiocchi. *Planning in Time*, in *Proceedings of the Conference on Temporal Aspects in Information Systems.* AFCET. France: May 1987, pp. 147-165.

Bjork, L.A., Jr. *Generalized Audit Trail Requirements and Concepts for Data Base Applications. IBM Systems Journal*, 14, No. 3 (1975), pp. 229-245.

Blanken, H. and A. Ijbema. *Language Concepts for Versioned Hierarchical Objects*, in *Proceedings of the Conference on Temporal Aspects in Information Systems.* AFCET. France: May 1987, pp. 191-207.

Brodie, M. *On Modelling Behavioral Semantics of Databases*, in *Proceedings of the Conference on Very Large Databases.* Cannes, France: Sep. 1981, pp. 32-42.

Casanova, M.A. and A.L. Furtado. *On the Description of Database Transition Constraints Using Temporal Languages*, in *Advances in Database Theory, Vol. II.* Ed. H. Gallaire, J. Minker and J.-M. Nicolas. New York: Plenum Press, 1984. Vol. II. pp. 211-236.

Ceri, S., G. Pelagatti and G. Bracchi. *Structured Methodology for Designing Static and Dynamic Aspects of Data Base Applications. Information Systems*, 6, No. 1 (1981), pp. 31-45.

Chou, H.-T. and W. Kim. *A Unifying Framework for Version Control in a CAD Environment*, in *Proceedings of the Twelfth International Conference on VLDB.* 1986.

Dietz, J.L.G. and K.M. van Hee. *A Framework for the Conceptual Modeling of Discrete Dynamic Systems*, in *Proceedings of the Conference on Temporal Aspects in Information Systems.* AFCET. France: May 1987, pp. 61-81.

Dittrich, K.R. and R.A. Lorie. *Version Support for Engineering Database Systems.* Technical Report RJ 4769. IBM. July 1985.

Easton, M.C. *Key-Sequence Data Sets on Indelible Storage.* Research Report RJ 4778 (50637). IBM Research Laboratory. July 1985.

Katz, R.H., M. Anwaruddin and E. Chang. *A Version Server for Computer-Aided Design Data*, in *23rd Design Automation Conference Proceedings.* ACM/IEEE. Las Vegas, NV: June 1986.

Kent, W. *Versions of Versioning.* Technical Report. Hewlett-Packard Laboratories. 1987.

Kimball, K.A. *The DATA System.* Master's Thesis, University of Pennsylvania, 1978.

Klahold, P., G. Schlageter and W. Wilkes. *A General Model for Version Management in Databases*, in *Proceedings of the Twelfth International Conference on VLDB.* 1986.

Lausen, G. *Analyse und Steuerung Paralleler Transaktionen in Einem Versionen-Datenbanksystem.* PhD. Diss. Karlsruhe, FRG, 1982.

LePape, C. and S.F. Smith. *Management of Temporal Constraints for Factory Scheduling*, in *Proceedings of the Conference on Temporal Aspects in Information Systems.* AFCET. France: May 1987, pp. 165-177.

Lindsay, B., L. Haas, C. Mohan, H. Pirahesh and P. Wilms. *A Snapshot Differential Refresh Algorithm*, in *Proceedings of ACM-SIGMOD.* Association for Computing Machinery. Washington DC: May 1986.

Lingat, J.Y., P. Nobecourt and C. Rolland. *Behaviour Management in Database Applications*, in *Proceedings of the Conference on Very Large Databases.* Ed. P. Hammersley. Brighton, England: Sep. 1987, pp. 185-196.

Lipeck, U.W. and G. Saake. *Monitoring Dynamic Integrity Constraints Based on Temporal Logic. Information Systems*, 12, No. 3 (1987), pp. 255-269.

McKenzie, E. and R. Snodgrass. *Scheme Evolution and the Relational Algebra.* Technical Report TR87-003. Computer Science Department, University of North Carolina at Chapel Hill. May 1987.

McKenzie, E. and R. Snodgrass. *Extending the Relational Algebra to Support Transaction Time*, in *Proceedings of ACM SIGMOD International Conference on Management of Data.* Ed. U. Dayal and I. Traiger. Association for Computing Machinery. San Francisco, CA: May 1987, pp. 467-478.

Mueller, T. and D. Steinbauer. *Eine Sprachschnittstele zur Versionenkontrolle in CAM-Datanbanken*, in Informatik-Fachberichte. Berlin: Springer, 1983. pp. 76-95.

Navathe, S.B. and R. Ahmed. *Temporal Aspects of Version Management. Database Engineering*, 7, No. 4, Dec. 1988, pp. 34-37.

Oberweis, A. and G. Lausen. *On the Representation of Temporal Knowledge in Office Systems*, in *Proceedings of the Conference on Temporal Aspects in Information Systems.* AFCET. France: May 1987, pp. 131-146.

Saake, G. and U.W. Lipeck. *Foundations of Temporal Integrity Monitoring*, in *Temporal Aspects in Information Systems*. Ed. C. Rolland, F. Bodart and M. Leonard. North-Holland, 1988. pp. 239-246.

Sarin, S.K., C.W. Kaufman and J.E. Somers. *Using History Information to Process Delayed Database Updates*, in *Proceedings of the Conference on Very Large Databases*. Ed. Y. Kambayashi. Kyoto, Japan: Aug. 1986, pp. 71-78.

Schueler, B. *Update Reconsidered*, in *Architecture and Models in Data Base Management Systems*. Ed. G. M. Nijssen. North Holland Publishing Co., 1977.

Verma, V. and H. Lu. *A New Approach to Version Management for Databases*, in *Proceedings of the AFIPS National Computer Conference*. Chicago, IL: AFIPS Press, June 1987, pp. 645-651.

Vianu, V. *Dynamic Constraints and Database Evolution*, in *Proceedings of the ACM Symposium on Principles of Database Systems*. Association for Computing Machinery. Atlanta, GA: Mar. 1983, pp. 389-399.

Weikum, G. *Entwurfsueberlegungen fuer Einen Versionen-Manager zur Realisierung eines Temporalen Datenbanksystems*. Technical Report DVSI-1983-A1. Darmstadt, FRG. 1983.

Zhenhe, G. and C. Kung. *On Temporal Aspect of Database Specifications. Sci. Sin. A. Math. Phys. Astron. Tech. Sci. (China)*, 30, No. 10, Oct. 1987, pp. 1102-12.

3 Valid Time

Valid time concerns the time when the information was valid in the real world. Relations that include valid time are termed *historical relations*.

Adiba, M.E. and N. Bui Quang. *Historical Multi-media Databases*, in *Proceedings of the Conference on Very Large Databases*. Ed. Y. Kambayashi. Kyoto, Japan: Aug. 1986, pp. 63-70.

Bassiouni, M.A. and M. Llewellyn. *Handling Time in Query Languages*, in *Proceedings of Statistical and Scientific Database Management Conference*. Rome: June 1988.

Blum, R.L. *Displaying Clinical Data from a Time-Oriented Database. Comput. Biol. Med.*, 11, No. 4 (1981), pp. 197-210.

Castillo, I.M.V., M.A. Casanova and A.L. Furtado. *A Temporal Framework for DataBases*, in *Proceedings of the Conference on Very Large Databases*. 1982.

Chaudhuri, S. *Temporal Relationships in Databases*, in *Proceedings of the Conference on Very Large Databases*. Los Angeles, California: Aug. 1988.

Chen, K. *The Inductive Acquisition of Temporal Knowledge*. Technical Report ISG 86-14. Department of Computer Science, University of Illinois at Urbana-Champaign. 1986.

Clifford, J. and A. Croker. *The Historical Relational Data Model (HRDM) and Algebra Based on Lifespans*, in *Proceedings of the International Conference on Data Engineering*. IEEE Computer Society. Los Angeles, CA: IEEE Computer Society Press, Feb. 1987, pp. 528-537.

Clifford, J. and A. Rao. *A Simple, General Structure for Temporal Domains*, in *Proceedings of the Conference on Temporal Aspects in Information Systems.* AFCET. France: May 1987, pp. 23-30.

Clifford, J. and A. Croker. *Objects in Time. Database Engineering*, 7, No. 4, Dec. 1988, pp. 11-18.

Date, C.J. *A Proposal for Adding Date and Time Support to SQL. ACM SIGMOD Record*, 17, No. 2, June 1988, pp. 53-76.

De, S., S. Pan and A. Whinston. *Temporal Semantics and Natural Language Processing in a Decision Support System. Information Systems*, 12, No. 1 (1987), pp. 29-47.

Ferg, S. *Modeling the Time Dimension in an Entity-Relationship Diagram*, in *4th International Conference on the Entity-Relationship Approach.* IEEE. Silver Spring, MD: Computer Society Press, 1985, pp. 280-286.

Fiadeiro, J. and A. Sernadas. *Behavioral Aspects of Intelligent Knowledge-based Information Systems*, in *Proceedings of the Conference on Temporal Aspects in Information Systems.* AFCET. France: May 1987, pp. 81-99.

Gadia, S.K. *The Role of Temporal Elements in Temporal Databases. Database Engineering*, 7, No. 4, Dec. 1988, pp. 19-25.

Gadia, S.K. *A Homogeneous Relational Model and Query Languages for Temporal Databases. ACM Transactions on Database Systems*, 13, No. 4, Dec. 1988.

Gadia, S.K. and C.-S. Yeung. *Inadequacy of Interval Time Stamps in Temporal Databases. To appear in Information Sciences*, (1988).

Garnic, D.K., A.T. Cohen and H.A. Sowisral. *Timestamping in Virtual Time Systems*, in *Proceedings of the Conference on Temporal Aspects in Information Systems.* AFCET. France: May 1987, pp. 225-239.

Jones, S. and P.J. Mason. *Handling the Time Dimension in a Data Base*, in *Proceedings of the International Conference on Data Bases.* Ed. S.M. Deen and P. Hammersley. British Computer Society. University of Aberdeen: Heyden, July 1980, pp. 65-83.

Lee, R.M. *A Denotational Semantics for Administrative Databases*, in *Proceedings of the IFIP WG 2.6 Working Conference on Data Semantics (DS-1).* Ed. T.B. Steel and R. Meersman. IFIP. Hasselt, Belgium: Jan. 1985, pp. 83-120.

Lorentzos, N.A. and R.G. Johnson. *TRA: A Model for a Temporal Relational Algebra*, in *Temporal Aspects in Information Systems.* Ed. C. Rolland, F. Bodart and M. Leonard. North Holland, 1987, pp. 203-215.

Lorentzos, N.A. and R.G. Johnson. *Extending Relational Algebra to Manipulate Temporal Data.* Internal Report NL/1/87. Department of Computer Science, Birkbeck College, London University. Aug. 1987.

Lorentzos, N.A. *A Formal Extension of the Relational Model for the Representation and Manipulation of Generic Intervals.* PhD. Diss. Birkbeck College, London University, Aug. 1988.

Lorentzos, N.A. and R.G. Johnson. *An Extension of the Relational Model to Support Generic Intervals*, in *Advances in Database Technology - EDBT'88.* Ed. J.W. Schmidt, S. Ceri and M. Missikoff. Springer-Verlag, 1988, pp. 528-542.

Lorentzos, N.A. and R.G. Johnson. *Requirements Specification for a Temporal Extension to the Relational Model. Database Engineering*, 7, No. 4, Dec. 1988, pp. 26-33.

Lorentzos, N.A. and R.G. Johnson. *Extending Relational Algebra to Manipulate Temporal Data. To appear in Information Systems*, 13, No. 3 (1988).

Lorentzos, N.A. and V.J. Kollias. *The Handling of Depth and Time Intervals in Soil Information Systems. To appear in Computers and Geosciences*, (1988).

McKenzie, E. and R. Snodgrass. *Supporting Valid Time: An Historical Algebra.* Technical Report TR87-008. Computer Science Department, University of North Carolina at Chapel Hill. Aug. 1987.

McKenzie, E. and R. Snodgrass. *An Evaluation of Historical Algebras.* Technical Report TR87-020. Computer Science Department, University of North Carolina at Chapel Hill. Oct. 1987.

Moens, M. *Temporal Databases and Natural Language*, in *Proceedings of the Conference on Temporal Aspects in Information Systems.* AFCET. France: May 1987, pp. 177-191.

Navathe, S.B. and R. Ahmed. *TSQL-A Language Interface for History Databases*, in *Proceedings of the Conference on Temporal Aspects in Information Systems.* AFCET. France: May 1987, pp. 113-128.

Navathe, S.B. and R. Amed. *A Temporal Relational Model and a Query Language. To appear in Information Sciences: An International Journal*, (1988).

Sadeghi, R. *A Database Query Language for Operations on Historical Data.* PhD. Diss. Dundee College of Technology, Dec. 1987.

Sadeghi, R., W.B. Samson and S.M. Deen. *HQL — A Historical Query Language.* Technical Report. Dundee College of Technology. Sep. 1987.

Sarda, N.L. *Modelling of Time and History Data in Database Systems*, in *Proceedings CIPS Congress 87 Winnipeg.* CIPS. May 1987, pp. 15-20.

Sarda, N.L. *Design of an Information System using a Historical Database Management System*, in *Proceedings of International Conference on Information Systems.* CIPS. Dec. 1987, pp. 86-96.

Sarda, N.L. *Algebra and Query Language for a Historical Data Model. To appear in The Computer Journal*, (1988).

Sarda, N.L. *Design of a Historical Database Management System.* Technical Report. Division of Math, Engr, and Computer Science, University of New Brunswick, Saint John, N. B. Canada. 1988.

Studer, R. *A Conceptual Model for Time*, in *Proceedings of the Sixth International Conference on Entity-Relationship Approach.* The ER Institute. New York, NY: Nov. 1987.

Tansel, A.U. *Adding Time Dimension to Relational Model and Extending Relational Algebra. Information Systems*, 11, No. 4 (1986), pp. 343-355.

Tansel, A.U. and M.E. Arkun. *HQUEL, A Query Language for Historical Relational Databases*, in *Proceedings of the Third International Workshop on Statistical and Scientific Databases.* July 1986, pp. 135-142.

Tansel, A.U. and M.E. Arkun. *Aggregation Operations in Historical Relational Databases*, in *Proceedings of the Third International Workshop on Statistical and Scientific Databases*. July 1986, pp. 116-121.

Tansel, A.U. *A Statistical Interface for Historical Relational Databases*, in *Proceedings of the International Conference on Data Engineering*. IEEE Computer Society. Los Angeles, CA: IEEE Computer Society Press, Feb. 1987, pp. 538-546.

Tansel, A.U. *A Historical Query Language. To appear in Information Sciences: An International Journal*, (1988).

Tansel, A.U. *Non First Normal Form Temporal Relational Model. Database Engineering*, 7, No. 4, Dec. 1988, pp. 46-52.

Tansel, A.U., M.E. Arkun and G. Ozsoyoglu. *Time-By-Example Query Language for Historical Databases. To appear in IEEE Transactions on Software Engineering*, (1989).

4 Both Transaction and Valid Time

Since transaction time and valid time are orthogonal aspects, it is possible to include both, resulting in a *temporal* relation.

Adiba, M.E., N. Bui Quang and J. Palazzo de Oliveira. *Time Concept in Generalized Data Bases*, in *ACM Annual Conference*. Association for Computing Machinery. Denver, Colorado: Oct. 1985, pp. 214-223.

Adiba, M.E. *Histories and Versions for Multimedia Complex Objects. Database Engineering*, 7, No. 4, Dec. 1988, pp. 3-10.

Ahn, I. and R. Snodgrass. *Partitioned Storage for Temporal Databases. Information Systems*, 13, No. 4 (1988).

Ahn, I. and R. Snodgrass. *Performance Analysis of Temporal Queries. To appear in Information Sciences*, , July 1989.

Ariav, G. *Handling The Time Dimension in ISs - A Research Agenda*. Technical Report 81-12-02. University of Pennsylvania. February 1982.

Ariav, G. and J. Clifford. *Temporal Data Management: Models and Systems*, in New Directions for Database Systems. Norwood, New Jersey: Ablex Publishing Corporation, 1986. Chap. 12. pp. 168-185.

Ariav, G. *Design Requirements for Temporally Oriented Information Systems*, in *Proceedings of the Conference on Temporal Aspects in Information Systems*. AFCET. France: May 1987, pp. 9-23.

Bui Quang, N. *Notion de Temps Dans les Bases de Donnees Generalisees*. Rapport de Dea Imag. IMAG Grenoble Univ France. June 1984.

Bui Quang, N. *Gestion des Historiques Pour la Base de Donnees Generalisees TIGRE*. R.R. TIGRE IMAG. IMAG Grenoble Univ France. June 1985.

Bui Quang, N. *Dynamic Aspects and Time Management in Generalized Database Systems*. PhD. Diss. Institut National Polytechnique de Grenoble, Nov. 1986.

Carmo, J. and A. Sernadas. *A Temporal Logic Framework for a Layered Approach to Systems Specification and Verification*, in *Proceedings of the Conference on Temporal Aspects in Information Systems.* AFCET. France: May 1987, pp. 31-47.

Chen, P.P.S. *The Time Dimension in the Entity-Relationship Model*, in *IFIP Information Processing 1986.* Ed. H.-J. Kugler. North-Holland, 1986.

Fagan, L.M. *VM: Representing Time-Dependent Relations in A Medical Setting.* PhD. Diss. Stanford University, June 1980.

Fugini, M.G., R. Maiocchi and R. Zicari. *Time Management in the Office-net System*, in *IFIP WG8.4 Workshop on Office Knowledge.* Toronto, Canada: Aug. 1987.

Gadia, S.K. and C.S. Yeung. *A Generalized Model for a Relational Temporal Database*, in *Proceedings of ACM SIGMOD International Conference on Management of Data.* Association for Computing Machinery. Chicago, Illinois: June 1988, pp. 251-259.

Gunadhi, H. and A. Segev. *Physical Design of Temporal Databases.* Technical Report LBL-24578. Lawrence Berkeley Lab. 1988.

Knolmayer, G. *Die Beruecksichtigung des Zeitbezugs von Daten bei der Gestaltung computer-gestuetzter Informationssysteme (Considering Time Aspects of Data in the Development of Computer-based Information Systems).* Technical Report 215. Institut fuer Betriebswirtschaftslehre, D-23 Kiel, Olshausenstrasse 40, Germany. May 1988.

Martin, N.G., S.B. Navathe and R. Ahmed. *Dealing with Temporal Schema Anomalies in History Databases*, in *Proceedings of the Conference on Very Large Databases.* Ed. P. Hammersley. Brighton, England: Sep. 1987, pp. 177-184.

McKenzie, E. *Bibliography: Temporal Databases. ACM SIGMOD Record*, 15, No. 4, Dec. 1986, pp. 40-52.

McKenzie, E. *An Algebraic Language for Query and Update of Temporal Databases.* PhD. Diss. Computer Science Department, University of North Carolina at Chapel Hill, Sep. 1988.

Rolland, C., F. Bodart and M. Leonard (eds). *Proceedings of the Conference on Temporal Aspects in Information Systems.* North Holland, May 1987.

Rotem, D. and A. Segev. *Physical Organization of Temporal Databases*, in *Proceedings of the International Conference on Data Engineering.* IEEE Computer Society. Los Angeles, CA: IEEE Computer Society Press, Feb. 1987, pp. 547-553.

Segev, A. and A. Shoshani. *Logical Modeling of Temporal Data*, in *Proceedings of the ACM SIGMOD Annual Conference on Management of Data.* Ed. U. Dayal and I. Traiger. Association for Computing Machinery. San Francisco, CA: ACM Press, May 1987, pp. 454-466.

Segev, A. and A. Shoshani. *Modeling Temporal Semantics*, in *Temporal Aspects in Information Systems.* Ed. F. Bodart C. Rolland, M. Leonard. North-Holland, 1988. pp. 47-58.

Segev, A. and A. Shoshani. *The Representation of a Temporal Data Model in the Relational Environment*, in *Proceeding of the 4th International Conference on Statistical and Scientific Database Management.* 1988.

Segev, A. and A. Shoshani. *Functionality of Temporal Data Models and Physical Design Implications. Database Engineering*, 7, No. 4, Dec. 1988, pp. 38-45.

Shoshani, A. and K. Kawagoe. *Temporal Data Management*, in *Proceedings of the Conference on Very Large Databases.* Kyoto, Japan: Aug. 1986, pp. 79-88.

Snodgrass, R. *The Temporal Query Language TQuel. ACM Transactions on Database Systems*, 12, No. 2, June 1987, pp. 247-298.

Snodgrass, R., S. Gomez and E. McKenzie. *Aggregates in the Temporal Query Language TQuel.* TempIS Technical Report 16. Computer Science Department, University of North Carolina at Chapel Hill. July 1987.

Snodgrass, R. (ed.) *Research Concerning Time in Databases: Project Summaries. ACM SIGMOD Record*, 15, No. 4, Dec. 1986, pp. 19-39.

Tasker, D. *An Entity-Relationship View of Time*, in *Proceedings of the Sixth International Conference on Entity-Relationship Approach.* The ER Institute. New York, NY: Nov. 1987.

Thirumalai, S. and S. Krishna. *Data Organization for Temporal Databases.* Technical Report. Raman Research Institute. 1988.

IEEE TRANSACTIONS ON

KNOWLEDGE AND DATA ENGINEERING

PREMIERE ISSUE MARCH 1989 RESERVE YOURS TODAY!

MARCH 1989 VOLUME 1 NUMBER 1 PREMIER ISSUE

AREAS ADDRESSED

- Aquiring and managing knowledge and data in developing and using information systems
- Strategies for efficiently capturing and storing new knowledge and data
- System modeling, design, access, security and integrity control mechanisms
- Structures in centralized and distributed information systems providing increased intelligence and ease of use through multiple interface devices
- Methods to prolong the useful life of knowledge and data and its graceful degradation

AREAS RECEIVING REGULAR COVERAGE

- AI techniques and expert systems
- Knowledge and data engineering tools and techniques
- System architectures
- Algorithms and applications
- System integration and modeling
- Query, design, and implementation languages
- Integrity, security, and fault tolerance
- Parallel and Distributed Processing

GUIDELINES FOR SUBMITTING PAPERS AND PROPOSALS ON SPECIAL ISSUES

(1) For invited papers and proposals for special issues, send 6 copies to:

C.V. Ramamoorthy, Editor-in-Chief
Computer Science Division
University of California, Berkeley
Berkeley, CA 94720
(415) 642-4751, (415) 642-1898
ram@ernie.berkeley.edu

(2) For all other submissions, send 6 copies of manuscript, complete with illustrations, abstract, and index terms, to:

Benjamin W. Wah, Associate Editor-in-Chief
Coordinated Science Laboratory
University of Illinois at Urbana-Champaign
1101 West Springfield Avenue
Urbana, IL 61801
(217) 333-3516, (217) 244-7175
wah%aquinas@uxc.cso.uiuc.edu

IEEE copyright transfer form and general guidelines for submissions can be found in the January 1988 issue of IEEE TRANSACTIONS ON SOFTWARE ENGINEERING.

MAKE ME A CHARTER SUBSCRIBER!

Send me the premiere issue of TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING as the first issue of my subscription.

☐ I'm a member of an IEEE society, so bill me for the member rate of $10 for a year (four issues).
IEEE Member Number: _____

☐ I'm not a member of an IEEE society, so bill me for the professional courtesy rate of $18 a year (four issues).
The professional organization I belong to that qualifies me for this rate is: _____

☐ Send me more information about the guidelines for submitting papers to TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING

Name _____
Address _____
City _____
State _____ Postal Zone _____
Country _____

Send to: IEEE Computer Society
Circulation Department
10662 Los Vaqueros Circle
Los Alamitos, CA 90720

OFFER EXPIRES JANUARY 31, 1989.

DOOD89

Call for Papers

The First International Conference on Deductive and Object-Oriented Databases Kyoto, Japan, December 4–6, 1989

Sponsored by: Information Processing Society of Japan (IPSJ)
Advanced Software Technology & Mechatronics Research Institute of Kyoto (ASTEM RI/Kyoto)
Supported by: ECRC, ICOT, INRIA, MCC
In cooperation with (requested): ACM, IEEE (approved), IEICE, JSAI, JS^{3}T
Location: Conference Hall of the Science Center Bldg. in Kyoto Research Park

Aim:

Deductive databases and object-oriented databases are at the forefront of research in next generation intelligent database systems. Object-oriented programming and design methodologies hold great promise in reducing the complexity of very large software systems in such domains as computer-aided design and manufacturing, integrated office information systems, and artificial intelligence. Object-oriented database systems will enhance the programmer/user productivity of such systems. Research in deductive databases is aimed at discovering efficient schemes to uniformly represent assertions and deductive rules, and to respond to highly expressive queries against the knowledge base of assertions and rules, and to check for the validity of knowledge. As for logic programming, logic has provided the basis for this area of research and nowadays there are strong interactions between these two fields. Recently, research has been aimed at integrating the object-oriented paradigm and rule-based deduction to provide a single powerful framework for intelligent database systems.

The primary objective of this conference is to provide a common forum of technical discussions between researchers in deductive databases and object-oriented databases, thereby accelerating the integration of the technical outputs of these two areas of research. We expect the conference to be continued on an annual basis with sponsorship from major next-generation computer projects around the world, such as ECRC, ICOT, INRIA, and MCC.

Topics included:

The conference will address new developments in theoretical and practical aspects of deductive databases (DD), object-oriented databases (OOD), and the integration of these two disciplines. Papers are solicited which describe original and novel research within this framework. The following is a partial list of topics of interest.

- Integrating Logic and Object Paradigm
- Concurrent Logic/Object Programming
- Integrity Enforcement in DOOD
- Artificial Intelligence Techniques in DOOD
- Query Languages and Query Optimization
- Advanced User Interface to DOOD Systems
- Operational DOOD Systems
- Integration of DOOD with Programming Languages
- Formalization of the Object-Oriented Concepts
- Performance of DOOD Systems
- Applications of DOOD Systems
- Consistency Checking in DOOD

Instructions: Authors should submit five (5) copies of a full paper to one of Program Committee Chairpersons by May 1, 1989. Papers should be no longer than 20 typewritten (double spaced) pages in English. The Author Name(s) and Affiliation(s) should appear on the cover sheet.

Program Committee Chairpersons

[American]
Won KIM
MCC
3500 West Balcones Center Drive
Austin, Texas 78759
U.S.A.
Computer Mail: kim@mcc.com

[European]
Jean-Marie NICOLAS
ECRC
Arabellastr. 17
8000 Munich 81
FR Germany
Computer Mail: jmn@ecrcvax. uucp

[Far East]
Shojiro NISHIO
Dept. of Applied Mathematics and Physics
Faculty of Engineering
Kyoto University
Kyoto, 606 Japan
Telex: 54231115 ENG KU J

IEEE Computer Society
1730 Massachusetts Avenue, N W
Washington, DC 20036-1903