Modelling and Managing Time in Database Systems

D. H. O. LING AND D. A. BELL
Department of Information Systems, Institute of Informatics, University of Ulster at Jordanstown, Northern Ireland, BT37 0QB

This paper describes a new architecture for a system to handle temporal data, called Temporal Data Management System, TDMS. TDMS represents an approach to the modelling of temporal semantics which is pragmatic because it takes into account practical issues such as the time/space trade-offs. We consider the technical aspects of the system i.e. how time domain data is managed in TDMS. The design of temporal databases is not addressed here.

We introduce a conceptual framework which permits issues such as handling schema anomalies due to updates, query validation and correct semantics modelling to be handled in a consistent and usable manner. It incorporates the time values of each of three 'data constructs' organised in a hierarchy. This is combined at the implementation level with a well-established multi-database architecture to give a comprehensive and efficient temporal model which overcomes several weaknesses of current temporal models. Modest extensions to the SQL syntax to handle time are proposed and have been implemented.

Both tuple-stamping and value-stamping are provided by the integration of TDMS and the relational data model.

TDMS has been prototyped under the UNIX operating system using the C programming language. Presently, TDMS interfaces with the Ingres DBMS.

Received October 1989, revised March 1992

1. INTRODUCTION

This paper describes a new architecture for temporal data systems – where the time-varying nature of reality is correctly and appropriately modelled – called Temporal Data Management System, TDMS. TDMS represents a pragmatic approach to the modelling and handling of time-dependent data values taking into account the time/space trade-offs.

Most of the current data models maintain only the concept of 'current view' data; an existing data value is overwritten by a new incoming value during the process of an update. Thus information about past data can no longer be made available. Attributes whose values are taken from the time domain effectively add an additional dimension to the conventional data models.[26] This demands a new concept of modelling to cope with such temporal data. Furthermore data management systems are also required to provide the functionalities to support temporal data.

The layout of this paper is as follows:

TDMS is introduced and its modelling aspects are described in Section 2. The architecture of TDMS, which uses the multi-database technique for managing temporal data and handling schema anomalies, is discussed in Section 3. Section 4 studies the operational aspects of TDMS – what new operations should the query sub-system provide for handling time? The 'landmark' approach for efficient selective retrieval of historical data is the subject of section 5, and concluding remarks are presented in Section 6.

2. THE CONCEPTUAL FRAMEWORK OF TDMS

Basically TDMS can be thought of as a system which contains a collection of views containing data values 'current' at particular points over some period of time.[13] Corresponding values belonging to the same object are chained or linked together along the time axis. Data values normally possess a step-wise time function; i.e. the persistence of a data object value is assumed from the first time of its validity until the time when it is superseded by another value. Therefore the existing value is made an appendage to the new current value and time-stamped (physical time) only when a change of state in the 'reality' being modelled occurs.

It is important to make a distinction between updates to data which are carried out because of the change in the state in the real world (referred to as *progressive updates* in this paper) and those which are carried out because an error in the existing data has been detected (referred to as *corrective updates*). One of the main features in most current temporal models (e.g. see Refs. 29 and 10) is that they keep a record of progressive update operations only and ignore the recording of corrective update operations. Thus information about corrections which are salient to future references are lost.

Consider a patient attending a clinic, who was diagnosed as having an eczema and treated with steroids on Monday. A few days later his condition was worse. Upon careful examination it was confirmed that he should have been diagnosed as having tinea, and should have been treated with antifungal agents all the time. Now if a progressive update is performed on this database record then the historical data in the updated database is semantically inconsistent. It wrongly indicates that the condition of the patient has progressed from eczema to tinea. However if a corrective update is performed by overwriting the old value with the new corrected value this creates inconsistency in the database too. Here the database wrongly indicates that the condition was tinea all along and was being treated with antifungal agents. These misrepresentations could be important from some subsequent legal enquiry, for example. This shows that the correct modelling of temporal semantics is important. To be sound, the system should reflect to the user that the patient was being wrongly diagnosed (at his first visit to the clinic) as having eczema and was treated with steroids, where in fact he should have been diagnosed as having tinea and been treated with antifungal agents all along.

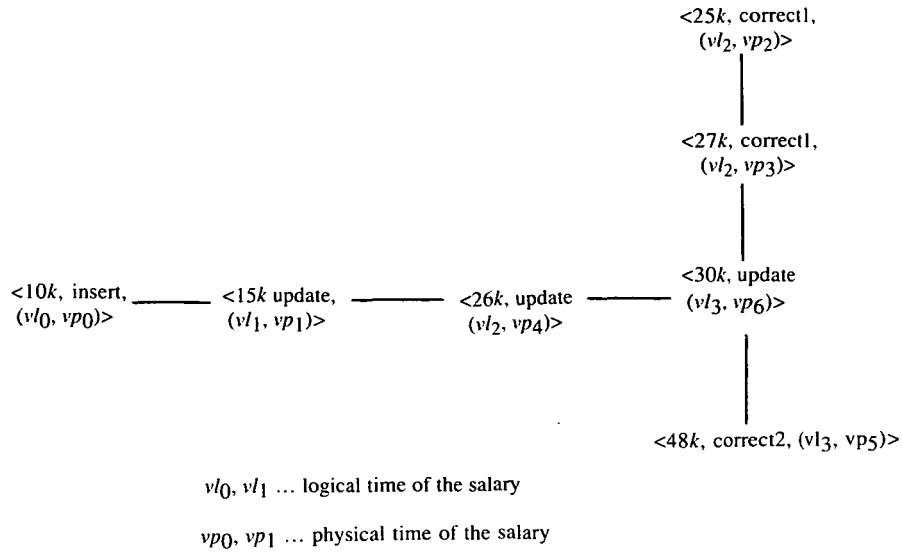

Fig. 1. A complete history of Tom's salary.

The conceptual framework presented below is rather elaborate, having been designed primarily for correctness in situations such as that above – its practical usability will be considered later. This complexity is unavoidable if the time modelling is to be useful, as has been discovered by many other researchers (e.g. see Ref. 32).

Temporal data in TDMS is represented by elementary data constucts which capture both the semantics and the data values of an object with respect to time. These elementary data constructs can be easily integrated with the relational data model. In this paper, we present a three-level hierarchy of data constructs for capturing temporal data using the relational data model. The highest level of construct in the hierarchy is called the relation level where the construct is used to identify a given database relation and its lifespan. The second level of the construct is the surrogate or tuple level which is used to identify a specific tuple. A time period is stored in this construct to indicate the time-wise validity of the tuple. The lowest level of construct is the value level which consists of attribute values for a given tuple, a status kind field (explained in detail later) and a time field which shows the time when the values become effective.

For clarity we start our discussion by considering a general data construct for capturing the semantics of any object, identified by a surrogate, s, and described by just one attribute.

(Note that when referring to a specific atomic object or value, the parameters in the data constructs are represented in lower-case letters. Upper-case letters are used to denote a collection of the atomic objects or values. This convention is adopted throughout this paper.)

This basic data construct is a 3-tuple in the form

$$\langle s, \{f\}, ot \rangle.$$

s is the surrogate of an object (i.e. a unique identifier), and ot shows logical (ol) value and physical (op) value time-pairs for both start and end of the object's life (4 values). f shows the history of the attribute using a 3-tuple, possibly repeating (denoted by { }), in the form $\langle v, k, vt \rangle$ where v is the attribute value, k is the status-kind (e.g. INSERT or UPDATE as described below), and vt = logical (vl) and physical (vp) values time-pair of the attribute.

$$\langle s, v_i, k_i, (vl_i, vp_i), (ol_i, op_i) \rangle, \quad 1 \leqslant i \leqslant n$$

with $ol_i = (vl_1, vp_1)$ for all i and $op_i = (vl_i, vp_i)$ for all i

This data construct can be considered at two levels; i.e. at the attribute level characterised by parameter f, and at the surrogate/tuple level. These two levels are explained in sections 2.1 and 2.2 below. Section 2.3 extends the hierarchy to the relation level.

2.1 Data construct for attribute value level

A general data construct for the attribute value level is represented as $f = \langle V, K, VT \rangle$ where $V \varepsilon \{v_1, v_2, v_3, \ldots, v_n\}$, $K \varepsilon \{k_1, k_2, k_3, \ldots, k_n\}$, and $VT \varepsilon \{(vl_1, vp_1), (vl_2, vp_2), (vl_3, vp_3), \ldots, (vl_n, vp_n)\}$.

As a simple example of how each of the 3-value tuples in F is used, consider the history of the salary of employee, Tom. A visual representation of this history is given in Fig. 1. We still consider just one attribute. In fact according to this basic model each attribute for an object s will have a similar construct.

Tom was first employed at (logical) time vl_0 and his salary (value) was $10k$. This information is recorded at (physical) time vp_0. This is represented in Fig. 1 by the tuple $\langle 10k, \text{insert}, (vl_0, vp_0) \rangle$. There are five (basic) possible values (i.e. INSERT, DELETE, UPDATE, CORRECT1, CORRECT2) one of which must be assigned to the status-kind field. When the value of an object is first created its status-kind is INSERT. When a value of an object is changed to another state its status-kind becomes UPDATE. When the object has terminated then its status-kind is DELETE. There are two possible types of corrective updates (Type 1 and Type 2) made to the database.

Type 1 correction (coded CORRECT1) is used to describe an error due to the inconsistency between the value stored in the computer and that of the reality. This may arise due to the mis-entry of data. A Type 2 correction (coded CORRECT2) is different from Type 1 in that the value stored in the computer and that of

the reality are identical but a mistake is made due to some wrong conclusion or decision in the real world.

In Fig. 1 the horizontal axis represents the event states having status of either INSERT, DELETE, or UPDATE. Type 1 corrections are represented by the upper half of the vertical axis, whereas the lower half of the vertical axis is used for recording Type 2 corrections. In order to improve the visual representations straight lines are drawn to link the event states together.

The progressive updates are presented on the horizontal axis.

Fig. 2 gives an illustration of the process of performing TYPE 1 correction in TDMS.

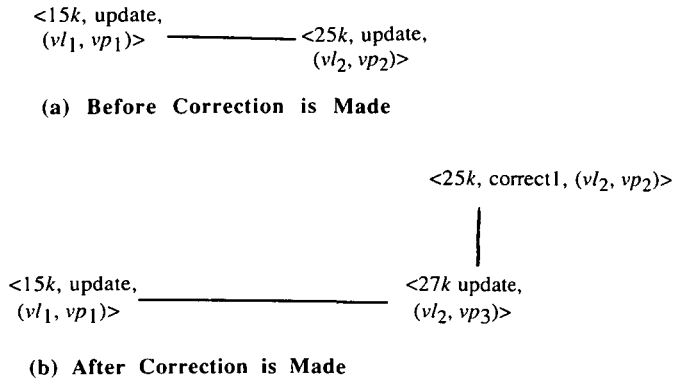

Fig. 2. Process of type 1 correction.

A question arises when examining Fig. 1 closely. What happens if there is a Type 1 correction detected at the first record having status of INSERT which marks the beginning of the object's lifetime? If its status field is overwritten by CORRECT1 then the event state having INSERT is no longer in existence. Given such an event state how could the system know if a given state is the beginning of the history under consideration? This problem is overcome by using the information stored in the data construct for the surrogate level. When an object is first created or inserted, a surrogate is uniquely assigned to the object. The logical and physical times of the first value stored for an attribute are identical to the start time of the surrogate which represents the object. Therefore checking an event state is done by comparing the logical and physical time-pairs between the surrogate and that given event state (i.e. as $ol_i = (vl_1, vp_1)$ holds for all i with $1 \leqslant i \leqslant n$ only $op_i := (vl_i, vp_i) = (vl_1, vp_1)$ has to be tested).

2.2 Data construct for surrogate/tuple level

The data construct for surrogate/tuple level is represented as $\langle S, \{F\}, OT\rangle$, as before, where $\{F\}$ is a collection of value states for object of surrogate S over time period OT. The time period OT gives the lifespan of the object. There are four time values stored in parameter OT; its first two time values are called the starting time points and its last two values are called the ending time points. The starting time points of the logical and the physical times are identical to the first event state stored. Similarly, the ending time points of the logical and the physical times are identical to the entry of the latest event state. This parameter OT has implications for temporal query validation and schema anomalies treatment. These issues will be further discussed later.

2.3 Data construct for relation level

In order to incorporate TDMS into the relational model using the value stamping method (Ref. 27) a new level of the data construct is introduced (data construct for the relation level) and is added onto the hierarchy of the basic data constructs.

The notion of Functional Dependency (FD) in TDMS can be expressed in the same way as in for conventional relational theory except that the conditions for FDs hold only when they are time-wise valid. Inference rules must take account of this (e.g. see Ref. 34). Also, the 'natural' key of a tuple (i.e. a minimal set of attributes that uniquely identify an entity in the usual sense) must be concatenated with its physical time values if it is to be referenced.

The data construct for the relation level is used to link together all the surrogates which belong to the same logical relation and it is represented as a 3-value tuples; $\langle r\_id, \{s\}, rt\rangle$ where r_id = relation identity, $\{s\}$ is a set of surrogates of the same logical relation and rt = logical and physical lifespan of the relation. The hierarchy of the data constructs of the relational TDMS is shown in Fig.

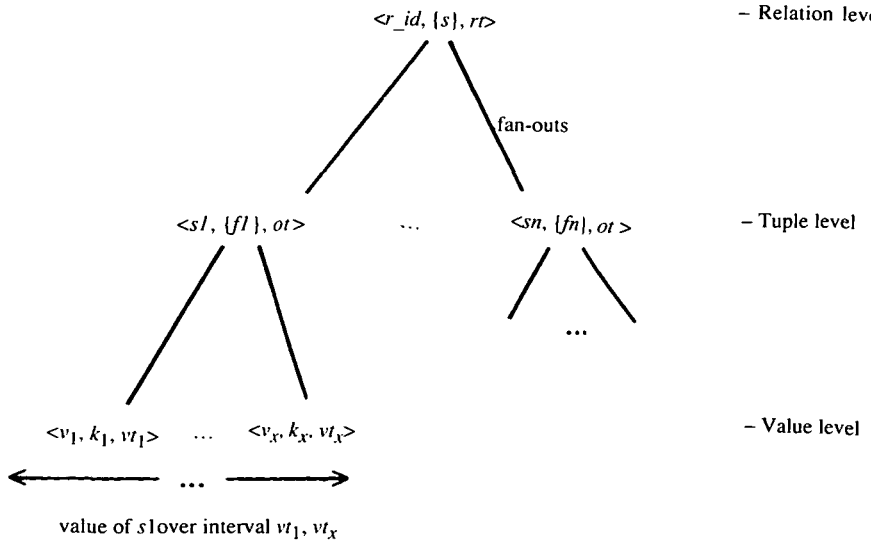

Fig. 3. Hierarchy of data constructs (value stamping).

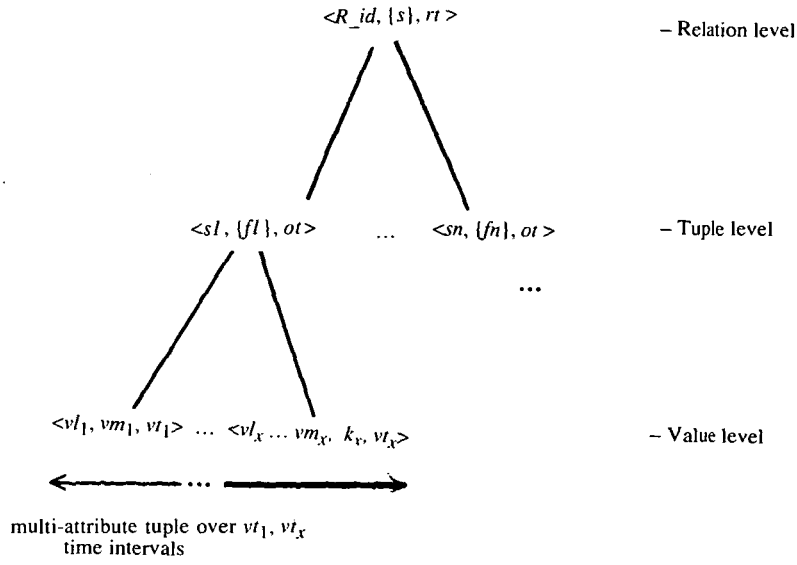

Fig. 4. Hierarchy of data constructs (tuple-stamping).

3. In Fig. 3 the fan-outs of $\{s\}$ are instances of data constructs for the surrogate level. Similarly, for example, the fan-outs of $\{f1\}$ are the set of data values belonging to surrogate $s1$ over time interval vt_1, vt_x.

This concept is similar to the time model found in Ref. 29 and is referred to as *value stamping*. It is suitable for relations with at most two attributes, though this time-stamping method can be efficient for retrieving the entire historical instances of a given object or surrogate, and is particularly useful for objects with high hit-rates.

The value stamping method found in Ref. 29 is limited to a key with at most one other attribute per relation. However a normal logical relation would consist of tuples with more than two attributes. In TDMS when a surrogate is taken to define a unique tuple which consists of multiple attributes then the parameter of $\{f\}$ in $\langle s, \{f\}, ot\rangle$ data construct contains a set of tuples defined over the time interval ot. A data construct at the relation level is needed to link together all the tuples of the same relation.

This hierarchy of data constructs is shown in Fig. 4. Physically the links between the data constructs can be accomplished by assigning pointers between them. This method of time-stamping is called tuple-stamping (Ref. 13).

In order to make tuple-stamping possible for TDMS, a new prefix value called 'PARTIAL_' is extended to the status-kind set. Semantically, this means that *some values in the tuples have been modified.* Therefore we can have $\langle s\_id, a_1, a_2, a_3, \ldots, a_m, k, vt\rangle$ where s_id = surrogate id of the tuple, $a_1, a_2, a_3, \ldots, a_m$ = the order values in the tuple (with m attributes), k = status-kind field (its values are INSERT, PARTIAL_UPDATE, PARTIAL_CORRECT1, PARTIAL_CORRECT2, and DELETE), and vt = logical and physical values time-pair of the surrogate. The underlying concept of TDMS clearly remains unchanged – the UPDATE, CORRECT1, and CORRECT2 values are merely prefixed by 'PARTIAL_'. With this extension TDMS model allows 'heterogeneous' time-stamping. Thus this leads to greater freedom in choosing efficient storage and performance trade-offs at the physical implementation design. This issue is discussed in section 5.

This hierarchical organisation of data constructs (either tuple or value stamping) can be directly mapped onto the system architecture of TDMS discussed below. The data construct for the attribute value level contains temporal data and hence is stored in the database. The other two higher data constructs are stored, managed and maintained by the management module(s) of the TDMS.

3. THE ARCHITECTURE OF TDMS

The use of the multi-database (MDB) concept for the architecture of the temporal model is introduced in this section. This architecture provides a neat way of dealing with temporal data handling and schema anomalies.

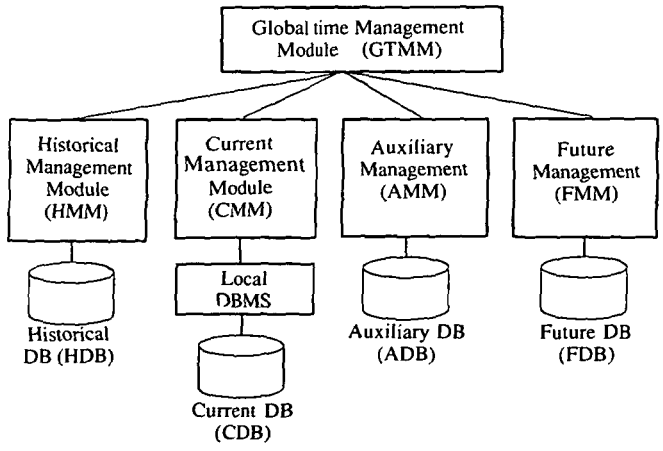

Fig. 5. The architecture of TDMS.

The architecture of TDMS shown in Fig. 5 is based on multi-database technique (see Refs. 3, 12 and 17). In a multi-database system, a global module is used to co-ordinate the sending/concatenating of the intermediate results to/from different underlying databases situated at different geographical sites. TDMS uses the same concept but treats underlying databases as data pertinent to different time periods. There are five main functional modules and three database types in TDMS. The local

DBMS and the current database are not part of TDMS but are included to provide a better understanding of the architecture. The local DBMS can be replaced by a flat file system, or it can be based on the relational or other data model. This local DBMS has the full responsibility of managing the 'current view' data. Note that in Fig. 5 separate databases (DBs) are used for storing the historical, auxiliary (Ref. 12) and future data. This approach improves efficiency of retrieval; each module is tailored to perform a specific function – thereby making temporal DBs easier to implement. Furthermore one could use different storage devices for different DBs. For example optical disks could be appropriate for storing non-volatile historical data, whereas auxiliary and future DBs can be retained on hard disks. The reasons for partitioning the HDB, CDB, FDB like this is as follows:

1. to relieve the schema anomalies problem – each of the 3 DBs could be governed by different DBMSs, and the MDB approach solves the potential problem of maintaining and integrating different schemas for different DBMSs;
2. to alleviate the integrity constraints problems – constraints on relations may vary over time and the MDB approach again handles this;
3. to reduce overall complexity – the general temporal functions are provided at the main control module and the underlying modules are dedicated to perform specific functions; and
4. to provide a distributed capability – placement of temporal data need not be confined to any dedicated computing facilities and hence overall computing costs can be optimised.

A temporal query is always posed to the Global Time Management Module (GTMM). The GTMM validates (if necessary), decomposes and optimises the temporal query into a set of equivalent subqueries for its underlying modules using conventional distributed database query handling techniques (e.g. Refs 5 and 4). Each underlying module will retrieve relevant data to form an intermediate result. In the particular case of the current management module (CMM) the subquery received from the GTMM may be further decomposed and optimised and translated into the local query language for the local DBMS. When data are required from both the current and the auxiliary DBs the sub-results have to be concatenated by the CMM. The intermediate results sent from all the underlying modules are concatenated in the GTMM to form a final result. This final result is then presented to the user.

The Current Management Module (CMM) is responsible for handling the 'current view' (i.e. the data here is a conventional database under a conventional DBMS, simply handling time as a normal attribute) temporal data which is beyond the capability of the local DBMS. For example, is easy to see that it would be very difficult to carry out required time period comparison in some DBMSs. In this case the 'current view' temporal data managed by the CMM is stored in the auxiliary database. For this reason only part of the CMM is dependent on the existing local data model. This approach makes TDMS flexible because current view data are not restricted to the local DBMS functions. The effort required to change only the dependent components of CMM is less than that to develop a full temporal model to be incorporated into the local existing DBMS. Normally the current commercially available DBMSs such as INGRES,[28] SQL/DS,[8] and DB2[9] allow some basic operations (such as comparisons) to be performed on temporal attributes. Therefore the dependent components of CMM for these DBMSs should be fairly small. In contrast, the PRECI_C prototype,[20] for example, handles dates as character strings. Here more effort is required to implement the dependent components of the CMM.

All historical data is transferred at the appropriate time from the current and/or auxiliary DBs into the Historical Database (HDB) and it is managed by the Historical Management Module (HMM). In this module tuples with the same surrogate are grouped together and are linked together. Operations given in the status–kind parameter are allowed to be applied to the HMM. This approach is described in section 2.1.

Similarly the Future Database stores the 'future view' temporal data and is managed by the Future Management Module (FMM). The Future DB contains all the future (logical time) data. Operations such as delete, insert, correct and update can be applied to the FMM. These operations follow the corresponding concepts given in section 2.1. When the future logical time of a value has in due course become current then this value and its associated data (e.g. correction(s) may have been made to this value before) are transferred to the CMM either by a Database Administrator (DBA) or automatically by the system. Details of FDB and its metamorphosis into a CDB are beyond the scope of this paper.

The Auxiliary DataBase (ADB) contains data which are not strictly under HMM or FMM components but are deemed necessary to the correct answering of queries. In general the ADB contains data for solving difficult problems such as null value treatment (Refs. 12 and 15) and database conflict (Ref. 31).

For example, in the medical world, hypotheses suggested during the diagnostic process often need to be retained and maintained by the system. The Auxiliary Management Module (AMM), is provided to facilitate this required kind of information processing.

When a temporal query involves retrieval of both historical and current data, the GTMM first of all validates the query and decomposes it into a set of equivalent temporal queries for the CMM and HMM. On receipt of its intermediate results from the underlying DBs, the GTMM integrates or concatenates these results, if appropriate, to form a final result.

The validity of the time period specified in a user query is established by checking against the *OT* and *RT* time values of the data constructs stored in the GTMM. A multi-database management system normally associates schema(s) with different geographical site(s) as nodes. In an analogous manner TDMS manages the data constructs and associates them with different temporal DBs as nodes. Using this approach user queries can be validated before any data access is made.

Schema update is not an uncommon phenomenon within the relational DB environment. An important question here is how to maintain consistency between the old and the new schema. The only research done on this area is found in Ref. 19. They proposed the following:

1. extend the two schemas each with a time-stamp to mark the validity of their time periods;
2. extend the SQL syntax to provide the facility of retrieving the previous schemas Ref. 19 by adding the PREV operator;
3. code extra SQL statements to merge the new and old schemas together (i.e. define a mapping between the old and new schemas);
4. embed these SQL statements into the system to be called up when needed.

This approach while sound seems to be somewhat naive and overly complicated. For example, the system has to maintain not only the schema differences but also the mapping between them using the SQL statements. If there are n schema versions then the query evaluation needs to navigate to the first version by specifying a 'PREV(1), PREV(2),..., PREV(n-1) current-schema-name' command.

In contrast, in TDMS such schema anomalies can be handled by duplicating the data construct at the relation-level (with the same *r_id*) and link the schemas (and databases) together using the multi-database concept. The old and new constructs each have valid lifespans and no additional attributes are needed to augment them. But TDMS requires that the old and new schemas have to be defined to let the system know where they are physically stored. A new special relation is also needed to define the old and new attributes of the relation shown in Fig. 6. Basically, it is a relation which contains two ordered tuples; the first tuple gives the information of the old relation and second gives the information of the current relational schema. An attribute column containing '_null' means that this attribute is not applicable. Corresponding attributes are appended under the same columns. This approach is simple and yet efficient for temporal query processing because no mapping language is needed and it requires minimum human intervention. For example a temporal query may request the retrieval of data under the old and current schemas. TDMS first performs the usual query validity checks. If there exist two data constructs for the same relation identifier (*r_id*) then appropriate subqueries can be formulated by looking up the special relation shown in Fig. 6. If a relation new_rel with NA1, OA3 and NA3 as its relational scheme (see Fig. 6) and if a global query in SQL form is

```
SELECT NA1, OA3
FROM new_rel
SINCE t3;
```

where the SINCE clause has an obvious meaning (explained later) then this query is decomposed (by a decomposer module in the GTMM) into an equivalent query such as

```
SELECT old_rel.OA1, old_rel.OA3, new_rel.NA1
       new_rel.OA3
FROM old_rel, new_rel
SINCE t3;
```

If both are located in different underlying DBs then further decomposition of the query by node would be required. The query decomposer module in a multi-database such as the EDDS prototype (Ref. 3) is suitable for temporal query decomposition but a slight extension to the module is required to handle the schema anomalies as mentioned above.

4. OPERATIONAL MODEL OF TDMS

The Operational Model provides a specification of the valid operations to be applied on the data constructs mentioned in the section 2. These operations facilitate access to and manipulation of the DB contents. In this paper, we aim only to outline these facilities.

The basic specification of an operation in TDMS has three generic parts directly corresponding to the three dimensions of the data construct (i.e. ⟨object, attribute, value, time⟩). The syntactic form of TDMS SQL is based on X/OPEN SQL[33] with necessary extensions for the specification of the temporal queries. These extensions are carefully formulated to ensure that no infringement can be made to the normal semantics and syntax of the SQL statements. The full extensions to the X/OPEN SQL syntax and more complex examples of the temporal queries are found in Ref. 16.

We shall describe how some of the temporal operations and query manipulation work by using some query examples.

To simplify the explanation in the following examples we require the following three assumptions

1. The granularity of time unit is one, and $t1, t2, \ldots, tn$ are time units with $t1 < t2, t2 < t3$, and so on;
2. The logical time column in a relation is considered as a special attribute in the semantic sense, though values are entered by an authorised user. Retrieval of the logical time column values must be specified in the query;
3. All tuples in the following two relations result from *progressive updates*:

Example 1

Salary relation

EMP	AMOUNT	LOGICAL from	to
John	10*k*	*t*2,	*t*6
John	11*k*	*t*6,	now
Lee	8*k*	*t*3,	*t*5
Lee	9*k*	*t*5,	*t*7
Ken	12*k*	*t*1,	now

Relation	Att_1	Att_2	Att_3	Att_4	Logical time	Physical time
old_rel	OA1	OA2	OA3	_null	*t*1, *t*5	*ti*
new_rel	NA1	_null	OA3	NA3	*t*5, now	*ty*

Fig. 6. Special relation for handling schema anomalies.

Example 1 (*cont.*)

Department relation

EMP	DEPT	LOGICAL from	to
John	shoes	$t4$,	$t8$
John	books	$t8$,	$t10$
Lee	shoes	$t3$,	$t6$
Lee	books	$t6$,	$t7$
Ken	shoes	$t1$,	now

Query: retrieve every employee's salary and department for his/her different assignments.
SQL: SELECT*

FROM *s* Salary, *d* Department
WHERE (*s*.EMP = *d*.EMP) SINCE START OF *d*.EMP;

Result:

EMP	AMOUNT	DEPT	LOGICAL from	to
John	10*k*	shoes	$t4$,	$t6$
John	11*k*	shoes	$t6$,	$t8$
John	11*k*	books	$t8$,	$t10$
Lee	8*k*	shoes	$t3$,	$t5$
Lee	9*k*	shoes	$t5$,	$t6$
Lee	9*k*	books	$t6$,	$t7$
Ken	12*k*	shoes	$t1$,	now

The SINCE START OF statement is necessary in order to distinguish the temporal query from the normal SQL query. To compute this result, a *temporal join* operator is used. It is quite similar to the relational join operator except that when the join involves two logical time attributes the tuples in each relation belonging to the same surrogate are, first of all, 'harmonised' time-wise between the two logical times. For example, the logical time intervals for John in relation Salary are $t2$–$t6$-now whereas the time intervals in relation Department are $t4$–$t8$–$t10$, where $t10$ < now. These time intervals can be considered as value ranges and harmonisation involves selecting the intersection of the two ranges. In this case the harmonised ranges are $t4$–$t6$–$t8$–$t10$ which gives rise to three new tuples. Any tuples (from either of the participating relations) of the same surrogate (e.g. John) whose logical time intervals include any of the three harmonised intervals will be appended to the result. The logic of the algorithm to carry out this harmonisation is similar to that for harmonising 'pieces' in the piecewise uniform method used in estimation of join sizes in distributed query optimisation (Ref. 4).

Example 2

Query: Retrieve John's salary in the interval $t3$, $t9$.

SQL: SELECT AMOUNT, LOGICAL_TIME
FROM Salary
WHERE (EMP = John) DURING ($t3$–$t9$);

Result:

EMP	AMOUNT	LOGICAL from	to
John	10*k*	$t3$,	$t6$
John	11*k*	$t6$,	$t9$

This query is processed by two operators, namely, select and temporal select. The select retrieves all tuples in relation Salary belonging to John giving:

EMP	AMOUNT	LOGICAL from	to
John	10*k*	$t2$,	$t6$
John	11*k*	$t6$,	now

Logical deduction on the user's part is now required in this case to get an explicit answer to this query (given directly above).

The temporal select is then applied to this intermediate result – to change the starting and/or ending point of all the tuples which have 'lives' outside the time intervals specified in the query. This gives the final result as shown above.

Example 3

Query: Find the number of employees ever employed by the company.
SQL: SELECT COUNT_UNIQUE(EMP)
FROM Salary
SINCE START OF EMP;

Result:

COUNT_UNIQUE(EMP)
3

This is a simple statistical query and is evaluated by projecting over EMP attribute then counting the number of rows existing. This number is the final result.

We complete our examples by giving a simple example of how corrective updates could effect the final result of a query. It should be pointed out that they do not affect any of the actual result's contents but rather provide the user with fuller information. Consider the following relation:

EMP_REL relation

name	salary	LOGICAL from	to	status
Tom	10*k*	$t1$,	$t6$	INSERT
Tom	12*k*	$t6$,	$t10$	CORRECT1
Tom	11*k*	$t6$,	now	UPDATE
Alex	13*k*	$t2$,	$t8$	INSERT
Alex	12*k*	$t8$,	$t10$	CORRECT2
Alex	14*k*	$t8$,	now	UPDATE

Question: Find the histories of all employees.
SQL: SELECT * FROM EMP_REL
SINCE START OF name;

Result:

name	salary	LOGICAL from	to
Tom	10*k*	*t*1,	*t*6
Tom	12*k* *1	*t*6,	*t*10
Tom	11*k*	*t*6,	now
Alex	13*k* *2	*t*2,	*t*8
Alex	12*k*	*t*8,	*t*10
Alex	14*k*	*t*8,	now

foot-note: *1 – tuple(s) *incorrectly entered* with logical-time = (*t*6, *t*10);
*2 – *external domain related error* with logical-time = (*t*8, *t*10);

The query processing is consistent with other normal temporal queries except that an indicator is tagged to the attribute value and an explanation of the indicator is displayed as a foot-note to the user. This may be helpful for ending some serious dispute over whether the mistake lies in the mis-entry or lies in the conclusion made in reality. This is obviously important to, for example, a computer-based hospital information system where the drug given to a patient (resulting in fatality) on a given date is due to mis-entry into the computer or wrongly prescribed by the medical doctor.

5. PHYSICAL MODEL OF TDMS

Time and memory are inseparable concepts. Storage and retrieval of temporal data are therefore important considerations in temporal databases. The behaviour of historical data growth can be categorised according to the frequency of the *hit-rate* and its corresponding *update-rate*. The hit-rate of a given attribute is said to be high if the attribute values are frequency accessed and retrieved. Similarly, if the update-rate of a given attribute is high then its attribute values are volatile and undergo frequent changes. In TDMS the storage structures for temporal data are determined by the hit-rate and update-rate profile.

Ideally, for retrieval performance reasons, we aim to store every complete instance of DB relation with respect to time – this allows duplicates of data to be stored. In contrast, for space efficiency reasons, we aim to store only the differential data between instances. Hence we seek to develop a satisfactory solution to accomplish these two antagonistic objectives.

We believe that most previous researchers have not been pragmatic in that they have not distinguished between high and low hit-rate data objects when considering storage strategies for handling time-related data. In practice often only a small percentage (e.g. according to a PARETO distribution – the 80–20 rule, say) of the tuples/attributes account for a very high proportion of the accesses. TDMS takes account of this phenomena in its storage strategies.

The frequency of the hit-rate to the relation is normally established or estimated at the system analysis and design stages. Therefore, assuming that current DB instances are more frequently accessed than the older ones the physical storage structure is determined in TDMS by the following two default criteria (the DBA gives the details in local cases):

RULE 1: If the hit-rate of at least 50% of the attributes/tuples in a relation is high (– these are called volatile attributes/tuples) then store every DB relation as a complete instance;

RULE 2: Otherwise provided that the relation hit-rate is high the logical relation is sub-divided into two special relations; one for the key and static attributes/tuples (only differential of data between DB instances are stored) and one for the key and volatile attributes/tuples (these DB attribute values will be stored as a complete DB instance).

Rule 1 is straightforward in that every complete instance of the logical relation is stored and this may imply some duplication of the data values between instances. Rule 2 gives a hybrid solution to reconcile the two extreme hit-rates. Therefore when a complete tuple is required, a 'time-merge' in the instance from both special relations of both hit-rates has to be performed. For example, a relation R with attributes A1, A2, A3 and A4, contains 3 instances (snapshots) of historical data. Attribute A1 is the key and A2 has a high hit-rate and high update-rate but A3 and A4 have a low hit-rate and low update-rate. Physically this relation has been divided into two special relations as depicted in Fig. 7 below.

A1	A2	A3	A4
X	*a*1	*c*1	*d*1
Y	*a*2	*c*2	*d*2
Z	*a*3	*c*3	*d*3

time = *t*0

A1	A2	A3	A4
X	*a*4	*c*1	*d*1
Y	*a*5	*c*2	*d*2
Z	*a*6	*c*3	*d*3

time = *t*1

A1	A2	A3	A4
X	*a*7	*c*1	*d*1
Y	*a*4	*c*2	*d*2
Z	*a*1	*c*3	*d*3

time = *t*2

(a) Logical Instances of Relation R

special relation1
(low hit-rate relation)

A1	A3	A4	Time	
X	*c*1	*d*1	*t*1,	*t*2
Y	*c*2	*d*2	*t*0,	*t*2
Z	*c*3	*d*3	*t*0,	*t*2

special relation2
(high hit-rate relation)

A1	A2	Time
X	*a*1	*t*0
Y	*a*2	*t*0
Z	*a*3	*t*0
X	*a*4	*t*1
Y	*a*5	*t*1
Z	*a*6	*t*1
X	*a*7	*t*2
Y	*a*4	*t*2
Z	*a*1	*t*2

(b) Physical Instances of Relation R

Fig. 7. Logical Vs physical instances of relation R.

The partition of such a relation would ensure the efficiency of storage and performance yet it is consistent with the logical data constructs of the model – as the data construct allows both the tuple and value time-stamping. The higher level data constructs themselves are indexed according to the primary tuple surrogates and a secondary index for object surrogates is maintained in the time sequence using a normal b^+-tree structure (which is used by typical DBMSs). The underlying files/records can now be indexed or hashed for efficient searching.

In terms of the temporal architecture of TDMS, each module maintains a subpart of the b^+-tree. For example, the top (towards the tree root) two levels of the b^+-tree are used to link the relation and surrogate data constructs and are maintained in the GTMM. The leaves of this portion of the b^+-tree are pointers which are set to point to its underlying sub modules for relevant data from the subtree. Physical access methods (e.g. using hash functions or indexing or b^+-tree) for the temporal data are analysed in Ref. 23. Efficient access to data files are also well presented elsewhere in the literature such as Refs 24, 2, and 11. However, each data access method has its advantages and disadvantages and for detailed information readers are referred to the above four references.

For access efficiency TDMS organises the data file/records via indexing and hashing. When historical data become large indexing and hashing, though helpful, can be less efficient for certain frequently accessed historical data. Direct indexing on particular high hit-rate data can be organised using b^+-trees – this is called the *land-mark approach*. This approach by-passes the normal hashing and indexing routes and the method is tailored for specific high hit-rate historical queries. For example, when less than 50% of the data in a relation have high hit-rates and regardless of the frequency of the update-rate, RULE 2 is applied which uses two special relations to represent a logical relation. When some portions of these special relations are very frequently accessed by the end-users then the landmark approach, using direct indexing to these frequently accessed portions of the special relations, can be set up to facilitate fast and efficient retrieval of temporal data. This landmark approach is similar to Matsuo *et al*'s work found in Ref. 18 except that they use it for retrieving very large document DBs.

6. DISCUSSION AND CONCLUSION

The basic conceptual time model of TDMS is based on value-stamping which is similar to that found in Ref. 26. However the TDMS conceptual model overcomes the weakness of that model in terms of semantic modelling. The parameters in the data constructs are minimal but are sufficient to model time comprehensively in an information system. The status-kind field in the data constructs helps to capture the fundamental and essential time semantics. The data construct provides both physical and logical time values. The logical time is specified in terms of a time period – i.e. a start point and an end time point, which together signify the duration of validity of an episode. The data construct requires that every value, even the surrogate value, should be time-stamped because a static attribute (e.g. employee name) does not remain in the DB permanently. This is different from Clifford's model[6] where surrogate/key values are not time-stamped. However temporal models such as in Refs. 1, 30, 7 and 26 do apply time-stamps on surrogate values.

It has also been shown that the data constructs of TDMS can be incorporated into the relational model (note that similar work has been done by Shoshani and Kawagoe[26] but their model limits a given relation to have only two attribute columns – which we consider to be too restrictive in practice). In fact this extension allows time-stamping to be applied to both tuple and attribute value levels. This unique concept combines with the multi-database architecture to give a comprehensive and efficient temporal model which overcomes many weaknesses of currently developed temporal models. Issues such as schema updates, query validation and correct semantic modelling are neatly handled in a consistent manner. Dynamic transmission of the historical data between differing bases may not be as straightforward as it looks (this is similar to the handling of objects in active databases – e.g. Refs. 21 and 22). It involves time-wise integrity and referential checkings, and a complex triggering with locking mechanism to ensure that data are handled in a consistent and valid manner. This issue will be reported in our forthcoming paper.[14]

The modest extensions of the SQL syntax definitions to include temporal operators are simple and consistent with the temporal manipulation of the system using the temporal operators.

The mapping between the conceptual and physical models of TDMS are straightforward. The conceptual data constructs are directly represented and indexed and linked by the physical access methods. For performance reasons, we have devised different file structures according to the behaviour of the temporal data. Therefore the physical model of TDMS is pragmatic – flexible, but efficient in storage and at the same time efficient in performance.

The conceptual, operational and physical aspects of TDMS have been discussed in the light of other temporal models. Though the model is simple, it has overcome several outstanding problems faced by other temporal models. The full capability of TDMS's architecture can be fully appreciated by incorporating it into a distributed heterogeneous DBMS where *different* underlying pre-existing temporal or non-temporal data models may be used to represent data geographically distributed over a computer network. Such an integrated system and its

distributed temporal query optimisation technique are described in Ref. 16.

The modules of TDMS have been coded in the C programming language and the system runs under Unix operating system. Presently, it interfaces with the Ingres DBMS. In future TDMS will be used to investigate further problems such as temporal query optimisation, transaction processing, concurrency control, and integrity constraint handling. We also intend to extend TDMS for integrating with a heterogeneous distributed DBMS. Investigation work has also begun to enhance the temporal query processing by incorporating some deductive capability. A possible solution is to extend the Logic Query Language (LQL)[25] which is a combination of functional and logic programming languages.

Acknowledgement

This research was carried out with the partial support of European Economic Commission (EEC) Multi-Annual Programme (grant 773B).

REFERENCES

1. G. Ariav, A temporally oriented data model. *ACM Transactions on Database Systems* **11** (4), 499–527 (1986).
2. D. A. Bell and S. M. Deen, Key space compression and hashing in PRECI. *Computer* **25** (4) (1982).
3. D. A. Bell, J. B. Grimson and D. H. O. Ling, EDDS – a system to harmonise access to heterogenous databases on distributed micros and mainframes. *Journal of Information and Software Technology* **29** (7), 362–370 (1987).
4. D. A. Bell, D. H. O. Ling and S. McClean, Pragmatic estimation of join sizes and attribute correlations. *Proc. IEEE Int. Conf. on Data Engineering, LA, USA (Feb. 1989)*, pp. 76–84.
5. S. Ceri and G. Pelagatti, *Distributed Databases*. McGraw-Hill, Maidenhead (1987).
6. J. Clifford, Towards an algebra of historical relational databases. *Proc. ACM-SIGMOD Int. Conf. on Management of Data, Austin, USA (May 1985)*, pp. 247–265.
7. S. K. Gadia and J. H. Vaishnav, A query language for a homogeneous temporal database. *Proc. 4th ACM SIGACT-SIGMOD Symposium on Principles of Database Systems, Oregon, USA (Mar. 1985)*, pp. 51–56.
8. IBM, *SQL/DS for VSE: a Relational Data System for Application Development*. IBM Form No. G320-6590 (1981).
9. IBM, Special Issues on DB2. *IBM System Journal* **23** (2) (1984).
10. S. Jones and P. J. Mason, Handling the time dimension in a data base. *Proc. Int. Conf. on Data Bases, Aberdeen, UK (July 1980)*, pp. 65–83.
11. D. Knuth, *The Art of Computer Programming*. Vol. 3. Addison-Wesley, Reading, MA, USA (1973).
12. T. Landers and R. L. Rosenberg, An overview of MULTIBASE. In *Distributed Databases*, edited H. J. Schneider, pp. 154–183. North-Holland, Amsterdam (1982).
13. D. H. O. Ling, D. A. Bell and I. R. Young, Time domain support for the medical information systems. *Proc 7th Int. Congress on Medical Informatics Europe, Rome (Sep. 1987)*, pp. 545–551.
14. D. H. O. Ling and D. A. Bell, *Dealing With Dynamic Objects in Temporal Databases*. Working paper (1989).
15. D. H. O. Ling, *Null Value Handling In a Heterogeneous Distributed Database System*. University of Ulster at Jordanstown, UK, Working paper R2.5/2/UU (Oct. 1986).
16. D. H. O. Ling, Query execution and temporal support in distributed database systems. *Ph.D. Thesis*, Institute of Informatics, University of Ulster, UK (July 1988).
17. W. Litwin, A logical model of a distributed database. *Proc. 2nd Int. Seminar on Distributed Data Sharing Systems*. Amsterdam, North-Holland (1981).
18. F. Matsuo, S. Futamura and T. Shinohara, Efficient storage and retrieval of very large document databases. *Proc. 2nd Int. Conf. On Data Engineering, LA, USA (Feb. 1986)*, pp. 456–463.
19. N. G. Martin, S. B. Navathe and R. Ahmed, Dealing with temporal schema anomalies in history databases. *Proc. of 13th Int. Cong. on Very Large Data Bases, Brighton, UK (Sep. 1987)*, pp. 177–184.
20. *Preci_C DBMS Manual*. University of Aberdeen, UK (1985).
21. T. Risch, Monitoring database objects. *Proc. 15th Int. Conf. on Very Large Data Bases, Amsterdam, Holland (Aug. 1989)*, pp. 445–454.
22. A. Rosenthal, U. S. Chakravarthy, B. Blaustein and J. Blakely, Situation monitoring for active database. *Proc. 15th Int. Conf. on Very Large Data Bases, Amsterdam, Holland (Aug. 1989)*, pp. 455–464.
23. D. Rotem and A. Segev, Physical organisation of temporal data. *Proc. 3rd Int. Conf. On Data Engineering, LA, USA (Feb. 1989)*, pp. 547–553.
24. B. Salzberg, *File Structures – an Analytic Approach*. Prentice-Hall, Englewood Cliffs, NJ, USA (1988).
25. J. Shao, D. A. Bell and M. E. C. Hull, LQL: a unified language for deductive database systems. *Proc. of Int. Conf. on The Role of Artificial Intelligence in Databases and Information Systems*. North-Holland, Amsterdam (1988), pp. 264–278.
26. A. Shoashani and K. Kawagoe, Temporal data management. *Proc. 12th Int. Conf. on Very Large Data Bases, Kyoto, Japan (Aug. 1986)*, pp. 79–88.
27. R. Snodgrass and I. Ahn, A taxonomy of time in databases. *Proc. ACM-SIGMOD Int. Conf. on Management of Data, Austin, USA (May 1985)*, pp. 236–246.
28. M. Stonebraker, E. Wong, P. Kreps and G. Held, The design and implementation of INGRES. *ACM Transactions on Database Systems* **1** (3), 189–222 (1976).
29. A. Segev and A. Shoshani, Logical modelling of temporal data. *Proc. ACM-SIGMOD Int. Conf. on Management of Data, San Francisco (Dec. 1987)*, **16** (3), 454–466.
30. A. U. Tansel, An extension of relational algebra to handle time in relational databases. *Proc. ACM-SIGMOD Int. Conf. on Management of Data, Austin, USA (May 1985)*, pp. 247–265.
31. M. Taylor, Data integration and query decomposition in DDBs. *PhD Thesis*, University of Aberdeen (1985).
32. D. C. Tsichritzis and F. H. Lochovsky, *Data Models*. Prentice-Hall, Englewood Cliffs, NJ, USA (1982).
33. X/Open, *X/open Portability Guide*. Elsevier, Amsterdam (1987).
34. Yang Chao-Chih, *Relational Databases*. Prentice-Hall, Englewood Cliffs, NJ, USA (1986).