Algebra and Query Language for A Historical Data Model

N. L. SARDA*
Division of Mathematics, Engineering and Computer Science, University of New Brunswick, Saint John, N.B. Canada E2L 4L5

A 'cubic' view of database with representation by time-stamping of tuples has been largely favoured so far in the research on modelling of time in database systems. In this paper, we propose a 'state' oriented view of historical databases. The salient features of our proposal are its simplicity, closeness to and consistency with classical relational model, and efficient implementability. We propose an algebra for historical relations which contains classical as well as some new operators. The operators are simple to comprehend, unlike in other research proposals. We are also able to formulate a completeness criteria for the proposed model. Finally, we extend the popular SQL query language for use with historical databases. Again, the extensions are consistent with the simple basis of standard SQL. They are minimal in number, yet very powerful and expressive. We illustrate algebra operators and extended SQL with many examples.

Received June 1987, revised December 1987

1. INTRODUCTION

A data model provides concepts and constructs for modelling data processing requirements of real-world organizations. A database management system (DBMS) incorporates a data model and provides high-level facilities for storage, retrieval and maintenance of data.

Time is an important dimension in all real-world activities. Events and actions occur continuously over time, modifying current status and generating history. The classical data models, such as relational, network or entity-relationship model, do not provide explicit concepts or support for modelling of time and accessing history data. If databases are to model a real-world application, a DBMS must explicitly provide concepts and facilities for modelling of time and management of history data. In this paper, we propose a data model for capturing of the time dimension. We refer to a DBMS which provides support for time and history data as a Historical DBMS (HDBMS).

Realizing the need to support time in database systems, a large number of research efforts have been directed towards studying various aspects of this problem (see the bibliography in Ref. 4 and research project summaries in Ref. 8). A 'state-oriented' conceptualization and a semantic model was proposed by Clifford and Warren[2]. Here, a historical relation (i.e., a relation of classical relational model extended to record timing information) has tuples stamped with time instants. Later, Clifford and Tansel proposed two views of historical relational algebra where relations have attributes stamped with time instants or periods.[3] Their algebras contain a large number of operators of varying complexity. The 'cubic' conceptualization of a historical relation is proposed in Snodgrass and Ahn[9] and in Ariav.[1] Snodgrass and Ahn also bring out very succinctly the need for two measures of time called real-world time and system time. They propose a few (but incomplete) extensions to the QUEL query language. Ariav defines the selection and projection operators on cubic view which are quite 'cumbersome'.[1]

In our earlier paper,[5] we identified the main issues which need to be answered satisfactorily before designing a practical HDBMS. We outlined our approach to HDBMS that was characterized by

(i) concept of 'state' for every database object (whether entity type or relationship type); a state prevails over a period of time.
(ii) real-world time measure, which would be generally equal to system time in on-line/real-time systems (HDBMS would include provisions for accepting out-of-sequence transactions).
(iii) separation of history data from current data for efficient storage and retrieval; the separation would be mostly transparent to database users. A very important consequence of this decision is that the database designer needs to model application requirements based only on 'current' perspective.
(iv) selectable time granularity, and
(v) extension of established query languages such as SQL[10] for definition, manipulation and control of data.

We have applied the proposed concepts to a real-life application in all its completeness: schema design, storage structure decisions, retrieval queries and update queries. This study is reported in Ref. 6. The primary motivation for this exercise was to identify basic but adequate HDBMS support that is both effective and easy to implement.

In this paper, we present our model in more formal terms. We define the concept of a historical relation and propose an algebra for them. We endeavour to keep the concepts simple as well as consistent with the classical relational data model. We also strive to identify a basic set of relational operators. Finally, we present extensions to SQL that are expressive and at high level.

The paper is organized as follows. The representation of time in terms of instants and periods, and the basic temporal operators are defined in Section 2. Section 3 defines historical relations and relates them with standard relations. The example in section 4 is used in the rest of the paper for illustrative purposes. The algebra for historical relations is presented in section 5. A basic

* Current address: Department of Computer Science & Engineering, Indian Institute of Technology, Bombay 400076, India

set of operators, which includes the standard relational operators (selection, projection and cartesian product) and two new operators called 'expand' and 'contract', is presented first, followed by some high-level operators which are very useful for time-related querying of historical databases. Section 6 contains extensions to SQL. Finally, in section 7, we highlight salient features of our work and contrast them in details with other research proposals.

We use the common terminology as per the standard text.[11]

2. TIME: REPRESENTATION AND OPERATIONS

Time is measured using a clock of suitable granularity. Every 'tick' of the clock represents a time *instant*. The value of an instant is the number of ticks from the start of clock. Thus, as in Ref. 3, time is isomorphic to the natural numbers and the set Time is a linear order, i.e., given instants t_1 and t_2, we either have t_1 equals t_2, t_1 is less than t_2, or t_2 is less than t_1.

The 'current time' refers to the latest clock tick, and is denoted by NOW. Thus, NOW can be thought as a 'moving' time variable as in Ref. 2.

A *period* is a consecutive sequence of time instants. It is represented as $t_1..t_2$, where $t_1 < t_2$ and the period includes all time instants starting from t_1 up to but *not* including t_2.

A *null period* does not include any time instant. The period $t_1..t_1+1$ contains only one time instant. We can now define the following operations on instants and periods:

(i) make period (..): given instants t_1 and t_2,

$$t_1..t_2$$

constructs a period, which is null if $t_1 >= t_2$. Note that '..' is not commutative.

(ii) Combine periods (+): given periods p_1 and p_2,

$$p_3 = p_1 + p_2 = p_2 + p_1$$

$$= \begin{cases} \text{null, if } p_1 \text{ and } p_2 \text{ have no common instants, else} \\ t_1..t_2\text{, such that each instant in } p_3 \text{ is either in} \\ p_1 \text{ or in } p_2. \end{cases}$$

(iii) extract overlapping period(*): given periods p_1 and p_2,

$$p_3 = p_1 * p_2 = p_2 * p_1$$

$$= \begin{cases} \text{null, if } p_1 \text{ and } p_2 \text{ have no common instants, else} \\ t_1..t_2\text{, such that each instant in } p_3 \text{ is contained} \\ \text{in both } p_1 \text{ and } p_2. \end{cases}$$

The following operations produce boolean result:

(iv) Included ($\in$): given instant t and period p, $t \in p$ = true if t is included in p, false otherwise.

(v) Meet ($\|$): given periods p_1, p_2, $p_1 \| p_2$ = true when p_1 is $t_1..t_2$ and p_2 is $t_2..t_3$; false otherwise.

(vi) Equal (=): given periods p_1,p_2, $p_1 = p_2$ = true when both include same set of time instants.

(vii) Contains ($\subset$): given periods p_1,p_2, $p_1 \subset p_2$ = true if all instants in p_2 are also contained in p_1.

(viii) Overlaps ($\cup$): given periods p_1,p_2, $p_1 \cup p_2$ = true if there is an instant which is included in both p_1 and p_2.

(ix) Comparison operations on time instants: as time instants are linearly ordered, we can use the standard comparison operators (=, <, >, < =, etc.) on two time instants.

(x) We will denote the set-theoretic 'contains or equal to' operation as $\subseteq$. We will also use it for periods with same meaning.

3. HISTORICAL RELATIONS

A relation scheme R in the conventional relational data model is defined over a set of attributes. Semantically, we may regard each relation scheme to be representing some kind of database object, either of an entity type or a relationship type. The concept of time is not (at least, explicitly) associated with R. To take an example, EMP(<u>ENAME</u>, RANK, PROJECT, SALARY) defines a relation for storing data about employees. A tuple in the relationship EMP gives employee name, rank, project assigned and salary. ENAME (the underlined attribute) is the *key* of EMP.

A historical relation scheme $\bar{R}$ is also defined over a set of attributes X. $\bar{R}$ will essentially model a real-world entity or relationship type. However, the concept of time is now explicitly included.

A tuple in $\bar{R}$ contains, besides values of the attributes, a time period p, indicating that the attributes had (or, have) those values in the time period p. A tuple is regarded as 'current' if its period has the form t_1..NOW. A tuple represents 'history' if its period is $t_1..t_2$ and $t_2 <$ NOW.

A tuple in $\bar{R}$ can *not* have a null period (we assume that such tuples are automatically deleted from $\bar{R}$).

It may be noted that while a tuple in R represents a database object, a tuple in $\bar{R}$ represents state (as given by values of its attributes) of a database object during certain time period. Also, $\bar{R}$ can be viewed as an extended R, where the extension can be automatically provided by a historical database management system (HDBMS). In fact, $\bar{R}$ can be regarded as a conventional relation if we had a way to refer to its period attribute. We would then have the advantage of using conventional relational model facilities (e.g., relational algebra and calculus) with historical relations. HDBMS can provide this link in a very simple way. For example, consider,

$\overline{\text{EMP}}$ (ENAME, RANK, PROJECT, SALARY)

containing the tuple

⟨JOHN, programmer, LOTUS, 30000, $t_1..t_2$⟩

which states that employee JOHN was a programmer on project LOTUS and earning 30000 during the period $t_1..t_2$.

The historical relation $\overline{\text{EMP}}$ is then equivalent to the following conventional relation:

EMP (ENAME, RANK, PROJECT, SALARY, PERIOD)

Moreover, a conventional relation can be viewed as a historical relation with every tuple stamped with the period ($-\infty$..NOW).

Let r be a tuple in $\bar{R}$; we will, then, assume the following notational facilities (from HDBMS):

r. PERIOD	gives period in *r*,
r. START	gives starting instant in period of *r*,
r. END	gives ending instant in period of *r*.

The concept of key for a historical relation is also defined. Logically, the attribute (group A of $\bar{R}$ is its key if it has unique values across all tuples at any point in time. Specifically, if s and t are any two tuples in $\bar{R}$, then either

$$s.A \neq t.A$$

or

$$s.A = t.A \text{ \underline{and not} } s.\text{PERIOD} \cup t.\text{PERIOD}.$$

In fact, it should be easy to see that if A is key of $\bar{R}$, it is *not* equivalent to saying that (A, PERIOD) is key of R, where R is the conventional counterpart of $\bar{R}$. The new definition of key for historical relations as given above can be used to extend the concepts of functional and multivalued dependencies for historical relations so that normal forms and normalization procedures of conventional relational data model can be appropriately extended to historical database.

We define two 'time' relations that can be useful in formulating queries.

A 'period relation' covering a period $t_1..t_2$ consists of a single tuple $\langle t_1..t_2\rangle$ and is denoted as $\{\langle t_1..t_2\rangle\}$.

An 'instant relation' covering a period $t_1..t_2$ contains one tuple for each instant in $t_1..t_2$. Specifically, it will contain the tuples $\langle t_1..t_1+1\rangle$, $\langle t_1+1..t_1+2\rangle$ ... etc. up to $\langle t_2-1..t_2\rangle$. It will be denoted by $\{\langle\langle t_1..t_2\rangle\rangle\}$. Note that both of these are historical relations.

4. EXAMPLE

We give here an example of historical database. It would be used subsequently for illustrating use of algebraic operations and query languages.

The schema contains the following historical relations:

$\overline{\text{EMP}}$ (<u>ENAME</u>, RANK, PROJECT, SALARY)

$\overline{\text{LAB}}$ (<u>L#</u>, LOC, MGR)

$\overline{\text{PROJ}}$ (<u>PROJECT</u>, L#, LEADER)

A tuple in EMP such as ⟨JOHN, PROGRAMMER, LOTUS, 30000, 7/85..11/85⟩ indicates that during the period 7/85..11/85 (i.e., July 85 to November 85), JOHN worked as PROGRAMMER in the LOTUS project and earned 30000 salary. We assume 'month' as the granularity level for time and that HDBMS provides suitable mechanism to view time in the familiar 'month/year' format. Other relations can be similarly understood (LAB stands for laboratory, LOC for location and MGR for manager; L# is unique identifier for laboratories).

Let the following queries be of special interest:

Q1: During 1985, what ranks were held by John and what were the durations (in 1985) of those ranks.

Q2: List employees who worked on LOTUS project for the whole of 1985.

Q3: List employees who worked (during 1985) on projects associated with laboratories located in NEW YORK during 1985.

These queries are singled out to illustrate various aspects of querying on time. Figure 1 contains a sample database for the above schema definition.

5. HISTORICAL RELATIONAL ALGEBRA

We will use symbols $X,Y,...$ to denote a set of attributes, $\bar{R}_1,\bar{R}_2,...$ for historical relations, and $S_1,S_2,...$ for conventional relations. R_1 will denote the conventional counterpart of $\bar{R}_1$. Thus, if $\bar{R}_1(X)$, then $R_1(X,\text{PERIOD})$.

By providing a simple link like this between historical and conventional relations, we are able to use standard algebra and calculus with historical relations. We briefly consider the standard and basic relational operators first. These operators are complete but primitive. More useful and high-level operations are defined in Section 5.2.

$\overline{\text{EMP}}$

ENAME	RANK	PROJECT	SALARY	PERIOD
JOHN	PROGRAMMER	LOTUS	30000	07/84..02/85
JOHN	ANALYST	LOTUS	40000	02/85..06/85
JOHN	ANALYST	LOTUS	42000	06/85..02/86
JOHN	SYS-MGR	ADA	45000	02/86..NOW
JANE	PROGRAMMER	LOTUS	32000	11/84..04/85
JANE	PROGRAMMER	LOTUS	34000	08/85..NOW
SMITH	ANALYST	LOTUS	40000	09/85..04/86
SMITH	ANALYST	ADA	42000	04/86..NOW

$\overline{\text{LAB}}$

L#	LOC	MGR	PERIOD
L1	LONDON	ROGER	03/84..06/85
L1	NEW YORK	SAM	06/85..NOW
L2	NEW YORK	NANCY	08/84..05/85
L2	NEW YORK	TIM	05/85..NOW
L3	BOMBAY	RAO	01/86..NOW

$\overline{\text{PROJ}}$

PROJECT	L#	LEADER	PERIOD
DBMS	L1	LINDA	06/84..09/85
ADA	L2	JOHN	02/86..NOW
ADA	L2	BROWN	03/84..02/86
PSL	L3	DAVID	02/85..08/85
PSL	L2	CHRIS	08/85..NOW

Figure 1. Example database

5.1 Basic algebra operations

Projection (π)

$$S = \pi_Y(\bar{R}_1),\ X \subseteq Y$$

The result is not a historical relation as it does not contain time periods. The projection operation on historical relation can still be conceived as being performed by scanning once each tuple in $\bar{R}_1$. We may use PERIOD as a projection attribute as in

$$\pi_Y(\bar{R}_1),\ \{X,\text{PERIOD}\} \subseteq Y$$

Here, the result may or may not be a historical relation, depending on whether the PERIOD attribute is included in the projection. Also, arithmetic or other operations may be specified on the projected attributes.

Selection (σ)

Let $F(Y)$ represent a predicate on attributes in Y. Then,

$$\sigma_{F(Y)}(\bar{R}_1),\ X \subseteq Y$$

or

$$\sigma_{F(Y)}(\bar{R}_1),\ \{X,\ \text{PERIOD}\} \subseteq Y$$

produce result which is a historical relation, σ is also logically performed by one scan on the period relation.

Product (χ)

The cartesian product combines tuples of operand relations. When applied to historical relations, as in

$$\bar{R}_1 \chi \bar{R}_2$$

the result is *not* a historical relation, even though it does contain two period-valued attributes.

It would seem that any language supporting the basic relational operators would also be 'complete' for historical databases because of the simple way provided by HDBMS for viewing historical relations as conventional relations. However, a difficulty arises due to the representation of state of a database object over a period of time. We might need to answer queries about an object at given instants or at all instants during a given period. We will, therefore, need new operators by which state (as given by a tuple) over a time period can be mapped into states at each instant in that period and vice-versa. The two operators defined below are basic and, along with the standard relational algebra and set-theoretic operations they can be used to define the notation of *completeness* for query languages for historical databases. We assert here (without giving a proof) that the following new operators can be simulated using instant relations and GROUP-BY and COUNT (as in SQL) functions.

Expand ($\bar{e}$)

$$\bar{R}_1 = \bar{e}(\bar{R}_2)$$

$\bar{e}$ is a unary operator whose result is a history relation. If $\bar{R}_2$ contains a tuple $\langle x, t_1 .. t_2 \rangle$, $\bar{e}$ produces one tuple in $\bar{R}_1$ for each instant in $t_1 .. t_2$ and all having same x-value. Thus, $\langle x, t_1 .. t_1+1 \rangle, \langle x, t_1+1 .. t_1+2 \rangle \ldots$ will be obtained for $\bar{R}_1$.

Contract ($\bar{c}$)

$$\bar{R}_2 = \bar{c}(\bar{R}_1)$$

$\bar{c}$ basically performs inverse function of $\bar{e}$. It combines those tuples of $\bar{R}_1$ which have same attribute values but consecutive or overlapping time periods into a single tuple with period that includes periods of combined tuples. Thus, if s and t are tuples in $\bar{R}_1$ and p_1 and p_2 are periods in s and t, then s and t are merged provided

$$s.X = t.X \ \underline{\text{and}}\ (p_1 \| p_2 \text{ or } p_1 \cup p_2)$$

The new tuple will be $s.X$, $p_1 + p_2$. Note that $\bar{R}_1$ and $\bar{c}(\bar{e}(\bar{R}_1)$ would be equivalent with respect to states of database objects, but they may not be equal on tuple-by-tuple basis. Also, note that $\bar{c}(\bar{e}(\bar{R}_1)$ gives same result as $\bar{c}(\bar{R}_1)$.

We now consider the example queries of Section 4 to illustrate the use of conventional and new operators with historical relations and to bring out the basic nature of $\bar{e}$ and $\bar{c}$ operators. To keep queries simple, we do not explicitly indicate selection to filter out tuples with null periods.

Q1

Let p stand for 1/85..1/86 (i.e., the year of 1985). Let

$$\bar{T} = \pi_A(\sigma_F(\overline{\text{EMP}}))$$

where

$$A \text{ is RANK}, \overline{\text{EMP}}.\text{PERIOD} * p$$

and

F is $\overline{\text{EMP}}$.ENAME = 'JOHN' $\wedge$ $\overline{\text{EMP}}$.PERIOD $\cup$ p

$\bar{T}$ contains the required answer. However, $\bar{T}$ may contain tuples with same rank but consecutive time periods. It is necessary to merge them before outputting the answer.

We need to use $\bar{c}$ to achieve the merging as follows:

$$\bar{c}(\bar{T})$$

With respect to data in Fig. 1, $\bar{T}$ gives

PROGRAMMER	1/85..2/85
ANALYST	2/85..6/85
ANALYST	6/85..1/86

However, $\bar{c}(\bar{T})$ gives (more appropriately)

PROGRAMMER	1/85..2/85
ANALYST	2/85..1/86

Q2

With p as before, let

$$T1 = \pi_{\text{ENAME}}(\sigma_F(\overline{\text{EMP}}))$$

where

$$F \text{ is PROJECT} = \text{'LOTUS'} \wedge \text{PERIOD} \cup p.$$

$T1$ gives employees (JOHN, JANE and SMITH for data in Fig. 1) who were on project LOTUS during 1985. We could subtract from $T1$ those employees who worked on other projects during 1985. However, this would not filter out JANE (who was laid off for a few months in 1985) and SMITH (who was hired first-time during 1985). The ideal way to solve the problem is to use $\bar{c}$ to aggregate states over time and then check whether 1985 is inclued in the period associated with the state:

$$\overline{T1} = \pi_{\text{ENAME.PERIOD}}(\sigma_F(\overline{\text{EMP}}))$$
$$result = \pi_{\text{ENAME}}(\sigma_{\text{PERIOD}cp}(\bar{c}(\bar{T}1)))$$

It is possible to formulate this query using the instant relation for 1985, and product and division operators of relational algebra.

Q3

Although query for Q3 can be formulated as a single algebraic expression, we will formulate it in steps for easy understanding. We will also use natural join ($\bowtie$) to simplify expressions. p, as before, represents 1/85..1/86. Intermediate results for database in Fig. 1 are also indicated.

(i) Obtain New York labs and their durations in 1985:

$$\bar{T}_1 = \pi_{L\#,PERIOD*p}(\sigma_{LOC='NEWYORK' \wedge PERIOD \cup p}\overline{LAB}))$$

⇒	L1	6/85..1/86
	L2	1/85..5/85
	L2	5/85..1/86

(ii) get projects and their labs during 1985

$$\bar{T}_2 = \pi_{PROJECT.L\#.PERIOD*p}(\sigma_{PERIOD \cup p}(\overline{PROJ}))$$

⇒	DBMS	L1	1/85..9/85
	ADA	L2	1/85..1/86
	PSL	L3	2/85..8/85
	PSL	L2	8/85..1/86

(iii) Obtain projects from $\bar{T}2$ which were done at labs in $\bar{T}_1$ at same time

$$\bar{T}_3 = \pi_A(\sigma_F(\bar{T}_1 \underset{L\#}{\bowtie} \bar{T}_2)$$

where A is PROJECT, T_1.PERIOD* T_2.PERIOD and F is T_1.PERIOD ∪ T_2.PERIOD

⇒	DBMS	6/85..9/85
	ADA	1/85..5/86
	ADA	5/85..1/86
	PSL	8/85..1/86

(iv) in a way similar to (iii) above, obtain employees working on projects in $\bar{T}_3$ at the same time.

5.2 Extended algebra operations

We now define a number of new operators which simplify formulation of queries on historical databases. A significant difference between these and standard operators is that time values are adjusted in the results when extended operators are applied to historical relations. As a consequence, results of these operators are always valid historical relations.

Project-and-widen (ω)

$$R_2 = \omega_Y(\bar{R}_1),\ X \subseteq Y$$

ω is similar to π except that

(i) period is included in the result, and
(ii) if two tuples in result match in attribute values and their periods are overlapping or consecutive then the two tuples are replaced by one with a combined period. To be more specific, let

$$\bar{R}_3 = \pi_{Y.PERIOD}(\bar{R}_1)$$

Then, to obtain $\bar{R}_2$ above from this $\bar{R}_3$, we carry out the following:

if $s,t \in \bar{R}_3$ and $s.Y = t.Y$ and

(i) if s.PERIOD ∪ t.PERIOD then replace s and t by $\langle s.Y, s.\text{PERIOD} + t.\text{PERIOD}\rangle$
(ii) if s.PERIOD ∥ t.PERIOD then replace s and t by $\langle s.Y, s.\text{START}..t.\text{END}\rangle$

Note that ω can *not* be directly expressed in terms of π, σ, χ and the operators defined on time in section 2. The reason for this is that ω may combine 2 or *more* tuples on the basis of consecutive/overlapping periods. However, we can express it using $\bar{e}$ and $\bar{c}$ as follows:

$$\omega_Y(\bar{R}) = \bar{c}(\bar{e}(\pi_{Y.PERIOD}(\bar{R})))$$

Time-slice (τ)

$$\bar{R}_2 = \tau_p(\bar{R}_1)$$

The operator τ is useful for obtaining status of database objects during the time period p. It is defined as follows:

$$\tau_p(\bar{R}_1) = \{\langle t.X,\ t.\text{PERIOD}*p\rangle \mid t \in \bar{R}_1\}$$

It may be recalled that tuples with null period in the result are discarded. τ can also be expressed using π, σ, χ (between $\bar{R}_1$ and the period relation $\{\langle p\rangle\}$) and the period operation $*$ defined in Section 2.

Concurrent product ($\underset{c}{\chi}$)

$$\bar{R}_3 = \bar{R}_1 \underset{c}{\chi} \bar{R}_2$$

The concurrent product differs from cartesian product in that it pairs only those tuples that have overlapping periods. The period in the result gives the overlap. Thus, if $\bar{R}_1(X)$ and $\bar{R}_2(Y)$, then $\bar{R}_1 \underset{c}{\chi} \bar{R}_2 = \{\langle t.X,\ u.Y,$ $t.\text{PERIOD} * u.\text{PERIOD}\rangle \mid t \in \bar{R}_1$ and $u \in \bar{R}_2\}$. It is noted again that tuples with null periods are automatically discarded from the result. We can express $\underset{c}{\chi}$ in terms of standard π, σ and χ. It is also easy to see that $\underset{c}{\chi}$ is commutative.

We next reconsider the earlier queries to illustrate the use of extended operators and to demonstrate their effectiveness in formulating time-related queries.

Q1

To find ranks and their periods in 1985 for the employee JOHN, we first select on JOHN, then project on ranks and periods overlapping with 1985 and also perform widening:

$$\omega_{RANK.PERIOD*p}(\sigma_{ENAME='JOHN'}(\overline{EMP}))$$

where p, as before, is 1/85..1/86.

Q2

To find employees who worked on project LOTUS for entire 1985, we first select tuples having project LOTUS and period overlapping 1985, then project-and-widen on ENAME, and, finally, select those employees whose period on LOTUS contains whole of 1985:

$$\sigma_{PERIOD \subseteq p}(\omega_{ENAME}(\sigma_{PROJECT='LOTUS' \wedge PERIOD \cup p}(\overline{EMP})))$$

Note that PERIOD condition in inner σ can be dropped in view of the period condition in outer σ.

Q3

To relate those tuples of the three relations $\overline{\text{EMP}}$, LAB and PROJ which were 'concurrently current' during 1985, we use the concurrent product operator and select:

$$\sigma_F(\overline{\text{EMP}} \underset{c}{\chi} \overline{\text{LAB}} \underset{c}{\chi} \overline{\text{PROJ}})$$

where F is $\overline{\text{LAB}}.\text{L\#} = \overline{\text{PROJ}}.\text{L\#} \wedge$

$\overline{\text{PROJ}}.\text{PROJECT} = \overline{\text{ENAME}}.\text{PROJECT} \wedge$
$\overline{\text{LAB}}.\text{LOC} = \text{'NEW YORK'} \wedge$
$\text{PERIOD} \cup 1/85..1/86$

Note that PERIOD in F refers to period in the result of concurrent product operations (whose result is a historical relation). We now perform projection to obtain the required result. Thus, the complete query for Q3 would be

$$\omega_{\text{ENAME}}(\sigma_F(\overline{\text{EMP}} \underset{c}{\chi} \overline{\text{LAB}} \underset{c}{\chi} \overline{\text{PROJ}}))$$

Note: an optimizing query processor would perform selections on periods and on LOC before performing $\underset{c}{\chi}$ operations. We can define algebraic properties of the standard and extended relational operators to formulate basis for optimizations.[7] The requirement that result tuples with null periods must be discarded can also be exploited by query optimizer to perform early selections.

6. TIME-ORIENTED QUERY LANGUAGE

In this section, we propose extensions to the popular query language SQL so that historical databases can be effectively and conveniently queried by end-users. We will refer to the extended SQL as TSQL.

The following objectives were set in making the extensions:

(i) retain the basic framework of a SQL query: it is easy to envisage (at least, conceptually) execution of a SQL query as a projection (on attributes given in SELECT) on those tuples of the cartesian product of relations (given in FROM) which satisfy a condition (given in WHERE). We wish to retain this conceptual simplicity because the historical relations, at the level of representation, are simple extensions of conventional relations.

(ii) retain the flexibility of SQL whereby a user may formulate a query in different ways (e.g., nesting instead of multiple relations in FROM).

(iii) the extensions should be minimum and simple to implement.

In the previous section, we applied standard relational algebra operations to the historical relations. We also proposed some new and some extended operators. On the basis of their analysis, we can establish directions for extending SQL:

(*a*) The expand ($\bar{e}$) and contract ($\bar{c}$) operations are useful as well as basic. We must incorporate them in some suitable form. The project-and-widen (ω) operator is sufficiently general and can be provided as equally useful but more practical alternative to $\bar{e}$ and $\bar{c}$.

(*b*) The need for concurrent product $\underset{c}{\chi}$ would be very common for relating data across two or more relations. Although it is not fundamental, we can provide it in TSQL so that users are saved from the burden of writing lengthy predicates on time in the WHERE clause.

(*c*) The fundamental assumption that result tuples with null periods are automatically discarded will be built into TSQL (again, this would simplify writing WHERE clause).

(*d*) Finally, it would be necessary to provide in TSQL the various operations on time discussed in Section 2.

TSQL may be used for historical and standard relations. We assume that HDBMS, as proposed in Refs. 5 and 6, provides facilities for defining both historical and conventional relations. A TSQL query on only standard relations must have same form and meaning as a SQL query. If R is a historical relation, we permit in TSQL the following references to R (in FROM clause):

R
CURRENT (R): refers to only current tuples of R
HISTORY (R): refers to non-current tuples of R

The primary objective in providing explicit referencing to the current status data is to permit its efficient quering (as the current data is expected to be used more often than history data). The efficiency is achieved by certain storage structures proposed in our HDBMS.[6]

The project-and-widen (ω) operator is supported in two ways:

(i) the SELECT phrase implies ω: thus, not only duplicates are removed but result tuples with matching attribute values and overlapping/consecutive periods are merged before outputting attributes listed in SELECT.

(ii) A new clause
TIME GROUPING ON attribute-list

is provided to basically implement $\omega_{\text{attribute-list}}$. It is equivalent to GROUP BY with the difference that projection is performed on listed attributes and, then, tuples with same values for attributes and overlapping/consecutive periods are merged. This clause may be followed by HAVING for selecting (groups of) tuples based on some predicate.

The concurrent product ($\underset{c}{\chi}$) is indicated simply by the (optional) word CONCURRENT after FROM. Thus,

FROM CONCURRENT relation-list

(at least, conceptually) produces a concurrent product of all relations given in the relation-list. Recall that result of $\underset{c}{\chi}$ is a historical relation (i.e., it would have a single time period).

Finally, the operations on time (Section 2) are provided in a direct and consistent manner with some renaming for readability. Specifically,

(i) *R*.PERIOD, *R*.START, *R*.END can be used (as in Section 3) to refer to time values in tuple *R*,
(ii) The period operations (.., +, *) can be used both in SELECT and WHERE, and
(iii) The period comparison operations (renamed as follows) can be used in WHERE and HAVING:

$\in$ WITHIN
$\|$ MEETS
= SAMEAS
$\subset$ INCLUDES
$\cup$ OVERLAPS

Since the extensions are minor and straightforward, we do not give syntax for TSQL (it is largely same as SQL). We now consider a few examples to illustrate its use for the database and queries given in Section 4.

Q1

To obtain ranks and their durations in 1985 for JOHN:

```
SELECT  RANK, r.PERIOD * (1/85..1/86)
FROM    EMP r
WHERE   ENAME = 'JOHN'
```

There is no need to do selection of tuples with periods overlapping with 1985 as it is implied by '*' operator on periods. Note that SELECT implies project-and-widen. An alternative way to formulate this query is:

```
SELECT  RANK, PERIOD
FROM    CONCURRENT EMP, {⟨1/85..1/86⟩}
WHERE   ENAME = 'JOHN'
```

Here, {⟨1/85..1/86⟩} is a constant period relation. The concurrent product implied by FROM above contains a single period value (in each result tuple), which is projected in SELECT.

Note: The following two TSQL queries are not equivalent:

```
(i) SELECT *
    FROM EMP
    WHERE PERIOD OVERLAPS (1/85..1/86)
(ii) SELECT *
    FROM CONCURRENT EMP, {⟨1/85..1/86⟩}
```

The result of (i) will contain unaltered period values from the tuples of EMP, while in (ii), periods in the result tuples will be contained in 1/85..1/86. The operation in (i) corresponds to σ and the operation in (ii) corresponds to τ (i.e., time-slice). The query (ii) above is equivalent to

```
SELECT ENAME, RANK, PROJECT, SALARY,
  PERIOD * (1/85..1/86)
FROM EMP.
```

Q2

To list employees who worked on project LOTUS for entire 1985 (employees who were hired for a part of 1985 but otherwise worked entirely on LOTUS are *not* to be included):

```
SELECT ENAME
FROM EMP
WHERE PROJECT = 'LOTUS'
TIME GROUPING ON ENAME
HAVING PERIOD INCLUDES 1/85..1/86
```

The TIME GROUPING clause performs project-and-widen on ENAME, producing result that contains ENAME and PERIOD. The FROM CONCURRENT may be used (as in Q1) to make the query execution more efficient. The following variation

```
SELECT ENAME
FROM CONCURRENT EMP, {⟨1/85..1.86⟩}
GROUP BY ENAME
HAVING SET (PROJECT) = {'LOTUS'}
```

is not same as earlier, since it would list employees who were hired only for a part of 1985.

Q3

To list employees working on projects of labs located in NEW YORK during 1985:

```
SELECT ENAME
FROM CONCURRENT EMP r1,
                LAB r2,
                PROJ r3,
                {⟨1/85..1/86⟩}
WHERE LOC = 'NEW YORK' and
  r1.PROJECT = r3.PROJECT and
  r3.L# = r2.L#
```

The condition in WHERE turns the concurrent product into a 'concurrent join'. A possible variation is to put a condition on PERIOD in WHERE instead of including the constant period relation in FROM.

7. CONCLUSIONS

In this paper, we have proposed the concept of historical relation to capture the ubiquitous time dimension of real-world activities. It has both a natural connotation and a simple representation. While other research works in this field have conceptualized 'temporal relation' as a 'cube',[1,9] we define it simply as a set of states, where a state, represented by a tuple, prevails over a period of time. The representation chosen by us is a simple extension of a conventional relation. In fact, both conventional and historical relations can co-exist in a database. With a provision for referencing period values, we can use standard relational operators and query languages for historical relations. This is a considerable advantage for practical applications of our model.

The basic relational operators (π, σ, χ) along with set-theoretic operations have been used to define 'completeness criteria' for relational query languages.[11] We have defined two more basic operators called 'expand' and 'contract', which cannot be expressed in terms of the basic relational operators. We need these operators primarily due to period-oriented recording of states. By using the new operators, we can construct state of a database object at every instant of time. Clifford and Warren in Ref. 2, in fact, defined semantics for temporal data model where a historical relation depicts state at every instant of time. We, therefore, regard expand and contract as fundamental operations for defining completeness criteria.

While standard operators are applicable, they are not always practical and effective, because queries on time have some unique requirements. We have defined three new operators called project-and-widen, time-slice and concurrent product. They are high-level and very useful operations as demonstrated by many examples in this paper.

Using the framework established by a set of useful algebraic operators, we finally extend the popular query language SQL[10] so that it can be effectively used for querying historical databases. Extensions to SQL were guided by three important objectives: (i) the conceptual basis of SQL queries must be retained, (ii) extensions should retain the basic flexibility of SQL, and (iii) extensions be minimal.

The high-level operators have been incorporated in SQL in a simple and consistent way. We have also illustrated power and expressiveness of extended SQL by many examples.

Besides closeness to the standard relational model, our proposal has the additional advantage of efficient support in both representation and query execution. The basic operations are performed on tuple-by-tuple basis and there are no 'implied or hidden' scans of the relations. Queries can be optimized on the basis of algebraic properties of operators and physical storage structures.[7]

We now review our proposal with other important contributions in this area of research.

Snodgrass and Ahn[9] have succinctly brought out two measures of time called real-world and system time. They propose 4 types of historical databases depending on extent of support for these two time measures. We consider real-world time to be of primary concern as computer system is merely a tool to maintain databases. We expect the two time measures to be equal for most transactions. As pointed out in Ref. 6, although support is required for processing out-of-sequence transactions, such transactions are not isolated events and they require considerable and careful planning in real-world environment. The considerations would be application-specific.

Still, it is possible to include multiple time measures in our model by user-defined attributes. It is possible to extend query languages to provide basic support for external time-measures (as in Refs. 9 and 1). Snodgrass and Ahn[9] have suggested a few (basically, only for time-slicing) extensions to the query language QUEL.

In Ref. 3, two historical relational algebras have been presented. In Clifford's view, a historical relation is an unnormalized relation with attribute values stamped with time instants at which the values became effective. Attributes are classified into three types based on their time properties. A large number of concepts (e.g., many types of nulls, life-span of a relationa and a tuple) and operations are identified, but there is no effort to identify basic set of operators and how they may relate to standard relational operators. Many types of selections, time-slicing operations and variety of joins make the picture quite complex. It is also difficult to efficiently support such a model, both in terms of storage and query execution.

In Tansel's view,[3] a historical relation is one with attributes stamped by time periods. Four types of attributes and a large number of operators are defined.

The 'cube' oriented conceptualization is proposed in the work of Ariav.[1] A tuple is extended in the time dimension by stamping it with the instant when it became current. Thus, a tuple, more appropriately, represents an event. In such models, a tuple is not really an independent element (as required in set-oriented definition of a relation) because another tuple indicates up to what time the state prevailed.

Ariav defines projection and selection operations on cubic view of historical relations. Both their conceptualization and practical realization are complex (for example, projection is not a simple tuple-by-tuple operation; it is necessary to check *sequences* of attribute values for removing duplicates). The SQL query language has also been extended in Ref. 1, but only with respect to projection and selection operations. However, some of the extensions do not fit into the simple basis of SQL queries (for example, WHILE clause or keyword EVERYWHEN require implicitly another scan of the source relation). Although some extensions proposed in Ref. 1 are useful and of very-high-level nature, their complex nature makes it difficult to comprehend them in the simple framework of standard SQL.

The above discussion indicates that the model proposed in this paper is simple, close to and consistent with the relational data model, and efficiently implementable.

Acknowledgement

I am grateful to the unknown referee for his/her valuable comments and suggestions to improve the paper.

REFERENCES

1. G. A. Ariav, Temporally oriented data model. *ACM Trans. Database Syst.* **11** (4), 499–527 (1986).
2. J. Clifford and D. S. Warren, Formal semantics for time in databases. *ACM Trans. Database Syst.* **6** (2), 123–147 (1983).
3. J. Clifford and A. V. Tansel, On an algebra for historical relational databases: two views. *Proc. ACM SIGMOD*, pp. 247–265 (1985).
4. E. McKenzie, Bibliography: temporal databases. *ACM SIGMOD RECORD* **15** (4), 40–51 (1986).
5. N. L. Sarda, Modelling of time and history data in database systems. *Proc. CIPS Congress 1987.* Winnipeg, pp. 15–20 (1987).
6. N. L. Sarda, Design of an information system using a historical database management system. *Proceedings of Intl. Conf. on Information Systems.* Pittsburg, pp. 86–96 (1987).
7. N. L. Sarda, Storage structures and query optimization for historical databases. (Under preparation).
8. R. Snodgrass (ed.), Research concerning time in databases: *Project Summaries.* ACM SIGMOD RECORD **15** (4), 19–39 (1986).
9. R. Snodgrass and I. Ahn, Temporal databases. *IEEE COMPUTER*, pp. 35–42 (1986).
10. D. D. Chamberlin *et al.*, SEQUEL 2: A unified approach to data definition, manipulation and control. *IBM Jour. Res. and Dev.* **20** (6), 560–575 (1976).
11. J. D. Ullman, Principles of database systems. *Computer Science Press* (2nd edition) (1984).