LBL-26417

Lawrence Berkeley Laboratory
UNIVERSITY OF CALIFORNIA

Information and Computing Sciences Division

To be presented at the 5th International Conference on Statistical and Scientific Database Management, Charlotte, NC, April 3–5, 1990, and to be published in the Proceedings

A Framework for Query Optimization in Temporal Databases

H. Gunadhi and A. Segev

November 1989

Prepared for the U.S. Department of Energy under Contract Number DE-AC03-76SF00098.

LBL-26417

A FRAMEWORK FOR QUERY OPTIMIZATION IN TEMPORAL DATABASES

LBL--26417

DE90 011339

Himawan Gunadhi & Arie Segev

Computing Science Research & Development
Information & Computing Sciences Division
Lawrence Berkeley Laboratory
1 Cyclotron Road
Berkeley, California 94720

and

Walter A. Haas School of Business
The University of California, Berkeley
Berkeley, California 94720

November 1989

Proceedings in the 5th International Conference on Statistical & Scientific Database Management, Charlotte, NC, April 3-5, 1990

This work was supported by the Director, Office of Energy Research, Applied Mathematical Sciences Research Program, of the U.S. Department of Energy under Contract No. DE-AC03-76SF00098.

LBL-26417
Dec. 1988

A FRAMEWORK FOR QUERY OPTIMIZATION IN TEMPORAL DATABASES

Himawan Gunadhi and Arie Segev

Walter A. Haas School of Business
The University of California and
Computing Sciences Research and Development Department
Lawrence Berkeley Laboratory
Berkeley, California 94720

Revised Nov. 1989

Abstract

We investigate issues pertaining to query processing of temporal databases in a relational environment. Tuple-versioning of relations is the adopted method of temporal data representation. New operators are necessary in order to exploit the richer semantics of temporal queries. We define four types of temporal joins-- theta-join, time intersection, time union and the event-join. Factors that affect processing strategies are discussed, especially the problem of estimating data selectivity for various temporal operations. Strategies for implementing the temporal equijoin operator are evaluated.

(Forthcoming in the 5th Int. Conf. on Statistical & Scientific Database Management)

This work was supported by the Applied Mathematical Sciences Research Program of the Office of Energy Research, U.S. Department of Energy under Contract DE-AC03-76SF00098.

1. INTRODUCTION AND MOTIVATION

The importance of temporal data models lies in their ability to capture the complexities of real world phenomena which are inherently dependent on time. Traditional approaches, such as the relational model of data, are incapable of handling all the nuances of such phenomena. Temporal models open up the possibility for new types of operations to enhance the retrieval power of a database management system (DBMS). One of the potential drawbacks of such models is the lack of processing efficiency-- the size of data and the complexity of time-oriented queries may yield unsatisfactory performance.

Many papers have been published on logical models that incorporate to varying degrees the time dimension. Most fall into the following categories: (1) Extensions to the relational model, e.g. [Clifford & Tansel 85, Ariav 86, , Clifford & Croker 87, Snodgrass 87]; (2) Enhancements of the Entity-Relationship model, e.g. [Klopproge & Lockemann 83, Adiba & Quang 86], and (3) Independent modeling such as the concept of the *Time Sequence Collection* (*TSC*) by [Shoshani & Kawagoe 86, Segev & Shoshani 87]. Many operators have been introduced in these papers, although in the relational context, the primary emphasis has been on their integration into the syntax of established query languages, such as SQL and QUEL. This is motivated by the desire to implement a temporal DBMS by minimal modification to current relational technology.

Our approach is to look into the functional requirements of queries on a *temporal relational database*. From there we define a set of fundamental join operators and investigate implementation and optimization strategies. We are motivated in part by the desire to study the feasibility of implementing the *TSC* model in relational form, or on top of an existing relational DBMS. In this paper, we do not attempt to define a complete set of temporal relational algebra, instead we focus on *temporal joins*, classified according to the attributes and operators specified in the join predicates. It is our belief that these joins should be capable of capturing the semantics of most, if not all, of the temporal join operators found in the literature. We outline several major issues that affect the design of query processing methods, with special emphasis on *selectivity estimation* of temporal relations for various operations. Finally, we look at a specific temporal operator, the *temporal equijoin*, and evaluate alternative strategies for its implementation.

The rest of the paper is organized as follows: In the next section, we discuss the relational representation of temporal data, and introduce basic definitions. In section 3, we define and discuss temporal operators and joins. In section 4, factors that impact query optimization are discussed, elaborating on mathematical modeling of the behavior of temporal relations. Implementation and efficiency issues pertaining to the temporal equijoin operator is explored in section 5. Finally, section 6 offers conclusions and an outline of future work.

Our contributions are:

- The classification and definition of four classes of temporal relational joins: Theta, Time Intersection, Time Union and Event joins. We feel that these definitions are needed for future research in the area of temporal query optimization.
- An introduction to the problem of selectivity measurement with respect to temporal relations, and how to model the dependencies that exist within such relations. As in traditional query optimization, deriving good selectivity estimates is of fundamental importance. In this paper we present the first step towards that goal.
- Evaluation of implementation strategies for the temporal equijoin, in the context of a relational environment. To the best of our knowledge this paper is the first to address the issue.

2. RELATIONAL REPRESENTATION AND DEFINITIONS

A convenient way to look at temporal data is through the concepts of *Time Sequence* (*TS*) and *Time Sequence Collection* (*TSC*) [Segev & Shoshani 87]. A *TS* represents a history of a temporal attribute(s) associated with a particular instance of an entity or a relationship. The entity or relationship is identified by a *surrogate* (or equivalently, the *time-invariant key*). For example, the salary history of an employee is a *TS*. A *TS* is characterized by several properties, such as the time granularity, lifespan, type, and interpolation rule to derive data values for non-stored time points. In this paper, for the sake of expositional convenience, we concentrate on one common type of data -- *stepwise constant* (*SWC*). *SWC* data represents a state variable whose values are determined by events and remains the same between events; the salary attribute represents *SWC* data. Time sequences of the same surrogate and attribute types can be grouped into a time sequence collection (*TSC*), e.g. the salary history of all employees forms a *TSC*. There are various ways to represent temporal data in the relational model; detailed discussion can be found in [Segev & Shoshani 88a]. We assume a time-interval representation, as shown in the examples of Table 1: The relations EMP_SAL, EMP_COM, EMP_MGR, EMP_DEP, DEP and DEP_TRAVEL represent employee salaries, commission rates of employees, employees' managers, employees' departments, department locations and departmental travel budgets respectively.

We use the terms *surrogate* (*S*), *temporal attribute*, and *time attribute* (T_S or T_E) when referring to attributes of a relation. For example, in Table 1, the surrogate of the EMP_SAL relation is E#, SAL is the temporal attribute, and T_S and T_E are time attributes. We assume that all relations are in first temporal normal form (1TNF) [Segev & Shoshani 88a]. 1TNF does not allow a surrogate instance to have more than one value of a temporal

EMP_SAL	E#	SAL	T_S	T_E
	E1	20	1	8
	E1	22	9	20
	E2	30	1	16
	E2	35	17	20
	E3	25	1	20

EMP_COM	E#C_RATE	T_S	T_E	
	E1	10%	1	7
	E1	12%	8	20
	E2	8%	2	7
	E2	10%	8	20

EMP_MGR	E#	MGR	T_S	T_E
	E1	TOM	1	5
	E1	MARK	9	12
	E1	JAY	13	20
	E2	RON	1	18
	E3	RON	1	20

EMP_DEP	E#	D#	T_S	T_E
	E1	D3	1	12
	E1	D2	13	20
	E2	D1	1	17
	E2	D2	18	20
	E3	D3	1	20

DEP	D#	FLOOR	T_S	T_E
	D1	4	1	20
	D2	1	1	7
	D2	2	8	20
	D3	2	1	7
	D3	5	8	20

DEP_TRAVEL	D#	BDGT	T_S	T_E
	D1	30	1	4
	D1	40	5	20
	D2	35	1	20
	D3	20	1	8
	D3	15	9	20

Table 1. Examples of Temporal Relations

attribute at a given time point. The implication for a temporal relation is that there are no two intersecting time intervals for a given surrogate instance. Whenever it is clear from the context, we will use the term "surrogate" instead of "surrogate instance". For the same reason, we often refer to the "temporal relation" as "relation".

2.1. Basic Notations

Let $r_i(R_i)$ be a relation on scheme $R_i = \{S_i, A_{i1}, ..., A_{im}, T_S, T_E\}$, where S_i is the surrogate of the relation with domain $dom(S_i)$, T_S and T_E are the time-start and time-end attributes respectively, with $dom(T_S) = dom(T_E)$. A_{ij} denotes the attribute with a corresponding domain $dom(A_{ij})$. We distinguish between the surrogate and other non-time attributes for expositional convenience. It is not necessary to distinguish between temporal and non-temporal A_{ij}'s, although one or more should be *time-varying* in order for temporal joins to produce non-trivial results. The characteristics and measures of the time attribute are described in [Segev & Shoshani 87]. It is assumed throughout that we are dealing with a time domain which can be represented as a finite or countably infinite set of integers.

Define $T_i = \{T_S, T_E\}$ as the *time-subscheme* and $R_i' = R_i - T_i$ as the *non-time subscheme* of r_i. Let x_i represent a tuple in r_i, and $x_i(\cdot)$ the projection of x_i on

some relational attribute(s). For a given tuple, $[x_i(T_S), x_i(T_E)]$ define a bounded interval, and the time-values immediately preceding and succeeding any of these boundaries are indicated by a decrement or increment of 1 respectively. Define r_1 and r_2 to be *T–compatible* if T_1 and T_2 are defined over compatible domains. Compatibility does not always mean identical domains, but we will assume so in this paper. The *time intersection* operator $x_1(T_1) \cap x_2(T_2)$ (or equivalently, x_1 *intersects* x_2) returns *true* if $x_1(T_S) \leq x_2(T_E) \wedge x_1(T_E) \geq x_2(T_S)$, and *null* otherwise, where r_1 and r_2 are *T–compatible*. We shall always assume that any joins on time are always made on *T–compatible* domains. Any *join* between r_1 and r_2 will produce r_3 with scheme $R_3 = R_1' \cup R_2' \cup T_3$, where the derivation of $r_3.T_S$ and $r_3.T_E$ which make up $r_3.T_3$ is dependent on the type of temporal join. Where *null* values are involved, we use $\varnothing$ to indicate the value for a single null attribute, and $\{\varnothing, ..., \varnothing\}$ for a set of such attributes.

3. TEMPORAL RELATIONAL OPERATORS

In this section we provide a description of temporal comparison operators and definition of temporal joins.

3.1. Temporal Comparison Operators

Comparisons over time attributes can be made at the explicit constraint level using standard arithmetic operators, i.e. "=", "≠", ">" and "≥", or at a higher level of semantics, for example "intersects" as defined previously. Many such operators have been defined in the literature [Allen 83, Navathe & Ahmed 86, Adiba & Quang 86, Segev & Shoshani 87]. The following is a list of the relevant ones:

x_1 *before* x_2 iff $x_1(T_E) < x_2(T_S)$

x_1 *overlaps* x_2 iff $x_1(T_S) < x_2(T_S) \wedge x_1(T_E) > x_2(T_S) \wedge x_1(T_E) < x_2(T_E)$

x_1 *starts* x_2 iff $x_1(T_S) = x_2(T_S) \wedge x_1(T_E) < x_2(T_S)$

x_1 *equal* x_2 iff $x_1(T_S) = x_2(T_S) \wedge x_1(T_E) = x_2(T_E)$

x_1 *during* x_2 iff $x_1(T_S) > x_2(T_S) \wedge x_1(T_E) < x_2(T_E)$

x_1 *finishes* x_2 iff $x_1(T_E) = x_2(T_E) \wedge x_1(T_S) > x_2(T_S)$

"Overlaps", "starts", "equal", "during" and "finishes" are subsets of "intersects"; they are defined in order to enhance the expressiveness of the query language. The predicate "before" can be more broadly defined as *t–before*, where $t \geq 0$, and measures units of time. This allows the predicate to be used to specify the *meet* ($t = 0$) and *precede* ($t = 1$) predicates, as well as represent arbitrary ordering relations, such as "x_1 2 units of time *before* x_2". Other temporal predicates not defined can be expressed in terms of conjunctions or disjunctions of the above set; e.g., the *disjoint* predicate can be expressed as "x_1 *before* x_2 or x_2 *before*

x_1".

3.2. Temporal Joins

A *temporal theta-join*, *T θ–join*, is made up of the *conjunction* of two sets of predicates, P_T and $P_{R'}$. P_T represents the set of time join predicates, i.e. those defined over time attributes, while $P_{R'}$ represents the set of non-time join predicates. There are three subclasses of temporal joins that are of special interest, based on the specification of join predicates: *Time intersection* class, *time union join* and *event–join*. Time intersection type of joins have a time predicate of $r_1.T_1 \cap r_2.T_2$. Where the non-time predicate has an equality operator, the join is called *temporal equijoin*, or *TE–join*, while if it is null, the join is a *time join* or *T–join*. In the event that the predicate is a non-equality type, we group it for processing purposes with the rest of the theta-join class. The semantics of a *TE–join* in the context of -1NF relations is given in [Clifford & Croker 87].

3.3. Temporal Equijoin

In the TE-join, two tuples $x_1 \in r_1$ and $x_2 \in r_2$ qualify for concatenation if the non-time join attributes have the same values and their time intervals intersect. Each concatenated tuple will have time attribute values that define the non-empty intersection of the two joining tuples. Note that the concatenation of tuples is non-standard, since only one pair of T_S and T_E attributes is part of the result tuples. If Y_{ij} are the non-time join attributes, where the subscripts i and j denote the relation number and attribute number respectively, then

$$r_1 \text{ TE-JOIN } r_2 \text{ on } Y_{11} = Y_{21} \wedge \cdots \wedge Y_{1m} = Y_{2m}$$

$$= \{x_3 | x_3(R_1') = x_1(R_1') \wedge$$

$$x_3(R_2') = x_2(R_2') \wedge$$

$$x_1(T_1) \cap x_2(T_2) \neq \varnothing \wedge$$

$$x_3(T_S) = max(x_1(T_S), x_2(T_S)) \wedge$$

$$x_3(T_E) = min(x_1(T_E), x_2(T_E))$$

$$\}$$

Given the query "Find the departments and their locations for all employees" on Table 1, we formulate the following join: EMP_DEP TE-JOIN DEP on EMP_DEP.D# = DEP.D#. The result is shown in Table 2.

Result	E#	D#	FLOOR	T_S	T_E
	E1	D3	2	1	7
	E1	D3	5	8	12
	E1	D2	2	13	20
	E2	D1	4	1	17
	E2	D2	2	18	20
	E3	D3	2	1	7
	E3	D3	5	8	20

Table 2. Result of TE-join between EMP_DEP and DEP relations

3.4. Time-Join

A T-join causes the concatenation of tuples from the operand relations only if their time intervals intersect. No predicate on non-time attributes is specified.

$$
\begin{aligned}
& r_1 \text{ T-JOIN } r_2 \\
&= \{x_3 | x_3(R_1') = x_1(R_1') \wedge \\
&\qquad x_3(R_2') = x_2(R_2') \wedge \\
&\qquad x_1(T_1) \cap x_2(T_2) \neq \varnothing \wedge \\
&\qquad x_3(T_S) = max(x_1(T_S), x_2(T_S)) \wedge \\
&\qquad x_3(T_E) = min(x_1(T_E), x_2(T_E)) \\
&\quad \}
\end{aligned}
$$

Although semantically a T-join is just a TE-join with a null predicate on the non-time attributes, it is a useful operator and is distinct from an optimization perspective. It is needed to answer the following query on the relations of Table 1: "Find employees who worked when at least one department had a travel budget greater than 38." The join is formulated as EMP_DEP T-JOIN DEP_TRAVEL, and the result shown in Table 3.

3.5. Time Union Join

The A TU-join is characterized by a union operation on the time intervals. There may be other time predicates specified, and we denote the set of such operators as $\bar{P}_T$. $P_{R'}$ can also be made of any arbitrary predicate. For every pair of tuples x_1 and x_2 that qualify on the other joining predicates, between one and three tuples can be produced, depending on the relationship between the time intervals of the operands. A TU-join is needed if a pair of tuples is considered to satisfy $P_{R'}$ even for cases where $x_1(T_1) \cap x_2(T_2) = \varnothing$. For example, the

Result	E#	D#	D#	BDGT	T_S	T_E
	E1	D3	D1	40	5	12
	E1	D2	D1	40	13	20
	E2	D1	D1	40	5	17
	E2	D2	D1	40	18	20
	E3	D3	D1	40	5	20

Table 3. Result of T-join between EMP_DEP and DEP_TRAVEL relations

following query requires a TU-join on the relations of Table 1: "Within the time interval [6,10], was any department's travel budget less than any employee's salary ?" (Note that the particular budget can be at a different time than the employee's salary.) We formulate the query as follows: DEP_TRAVEL TU-JOIN EMP_SAL on DEP_TRAVEL.BDGT < EMP_SAL.SAL, and the resulting relation is shown in Table 4. The union join operation is somewhat analogous to a cartesian product operator in the conventional database context.

Result	D#	BDGT	E#	SAL	T_S	T_E
	D3	20	Ø	Ø	6	8
	Ø	Ø	E1	22	9	10
	D3	20	E2	30	6	8
	Ø	Ø	E2	30	9	10
	D3	20	E2	35	6	8
	Ø	Ø	E2	35	9	10
	D3	20	E3	25	6	8
	Ø	Ø	E3	25	9	10
	Ø	Ø	E1	20	6	8
	D3	15	Ø	Ø	9	10
	D3	15	E1	22	9	10
	Ø	Ø	E2	30	6	8
	D3	15	E2	30	9	10
	Ø	Ø	E2	35	6	8
	D3	15	E2	35	9	10
	Ø	Ø	E3	25	6	8
	D3	15	E3	25	9	10

Table 4. Result of TU-join between DEP_TRAVEL and EMP_SAL

Formally,

$$r_1 \text{ TU-JOIN } r_2 \text{ on } P_{R'} \wedge \bar{P}_T = r_{31} \cup r_{32} \cup r_{33}$$

where

$$r_{31} = \{x_{31} | x_{31}(R_1\hat{}) = x_1(R_1\hat{}) \wedge$$

$$x_{31}(R_2\hat{}) = x_2(R_2\hat{}) \wedge$$

$$P_{R'} \;\&\; \bar{P}_T \wedge$$

$$x_1(T_1) \cap x_2(T_2) \neq \emptyset \wedge$$

$$x_{31}(T_S) = max(x_1(T_S), x_2(T_S)), \;\&\; x_{31}(T_E) = min(x_1(T_E), x_2(T_E))$$

$$\}$$

$$r_{32} = \{x_{32} | x_{32}(R_i\hat{}) = x_i(R_i\hat{}) \wedge$$

$$x_{32}(R_j\hat{}) = \{\emptyset, ..., \emptyset\} \wedge$$

$$P_{R'} \;\&\; \bar{P}_T \wedge$$

$$x_i(T_S) < x_j(T_S) \wedge$$

$$x_{32}(T_S) = x_i(T_S) \;\&\; x_{32}(T_E) = min(x_i(T_E), x_j(T_S) - 1)$$

$$i = 1 \text{ or } 2;\; j = 2 \text{ if } i = 1 \text{ and } j = 1 \text{ if } i = 2$$

$$\}$$

$$r_{33} = \{x_{33} | x_{33}(R_i\hat{}) = x_i(R_i\hat{}) \wedge$$

$$x_{33}(R_j\hat{}) = \{\emptyset, ..., \emptyset\} \wedge$$

$$P_{R'} \;\&\; \bar{P}_T \wedge$$

$$x_i(T_E) > x_j(T_E) \wedge$$

$$x_{33}(T_S) = max(x_i(T_S), x_j(T_E) + 1) \;\&\; x_{33}(T_E) = x_i(T_E)$$

$$i = 1 \text{ or } 2;\; j = 2 \text{ if } i = 1 \text{ and } j = 1 \text{ if } i = 2$$

$$\}$$

3.6. Event-Join

An event-join groups several temporal attributes of an entity into a single relation. This operation is extremely important because due to normalization, temporal attributes are likely to reside in separate relations. To illustrate this point, consider an employee relation in a non-temporal database. If the database is normalized we are likely to find all the attributes of the employee entity in a single relation. If we now define a subset of the attributes to be temporal (e.g., salary, department, manager, commission-rate, etc.) and they are stored in a single relation, a tuple will be created whenever an event affects at least one of those attributes.

Consequently, grouping temporal attributes into a single relation should be done if their event points are synchronized. Regardless of the nature of temporal attributes, however, a physical database design may lead to storing the temporal attributes of a given entity in several relations-- this is the case for the employee relations in Table 1. The analogy in a conventional database is that the database designer may create 3NF relations, but obviously, the user is allowed to join them and create an unnormalized result.

The event-join operation combines elements of the temporal equijoin and and time union join. In order to describe an event-join between r_1 and r_2, we first present the operator *TE–OUTERJOIN*. A TE-outerjoin is a directional operation from r_1 to r_2 (or vice versa). For a given tuple $x_1 \in r_1$, outerjoin tuples are generated for all points $t \in [x_1(T_S), x_1(T_E)]$ such that there does not exist $x_2 \in r_2$ with $x_2(S) = x_1(S)$ and $t \in [x_2(T_S), x_2(T_E)]$. Note that all consecutive points t that satisfy the above condition generate a single outerjoin tuple. Using those operations the event-join, r_1 EVENT-JOIN r_2, is done as: (1) temp1 $\leftarrow r_1$ TE-JOIN r_2 on S; (2) temp2 $\leftarrow r_1$ TE-OUTERJOIN r_2 on S; (3) temp3 $\leftarrow r_2$ TE-OUTERJOIN r_1 on S; (4) result $\leftarrow$ temp1 $\cup$ temp2 $\cup$ temp3. Given the query "Find the managers and commission rates received by employees", we formulate the following event-join query: EMP_MGR EVENT-JOIN EMP_COM. The result of this join is shown in Table 5.

Result	E#	MGR	C_RATE	T_S	T_E
	E1	TOM	Ø	1	1
	E1	TOM	10%	2	5
	E1	Ø	10%	6	7
	E1	Ø	12%	8	8
	E1	MARK	12%	9	12
	E1	JAY	12%	13	20
	E2	RON	Ø	1	1
	E2	RON	8%	2	7
	E2	RON	10%	8	18
	E2	Ø	10%	19	20
	E3	RON	Ø	1	20

Table 5: Result of Event-Join between EMP_MGR and EMP_COM

We can now provide a formal definition of an event-join. Let I denote an arbitrary interval $[T_S, T_E]$ over time; for two intervals I_1 and I_2, $I_1 \subseteq I_2$ if $I_1.T_S \geq I_2.T_S$ and $I_1.T_E \leq I_2.T_E$; the cardinality of an interval, $|I|$, is measured as $|T_E - T_S + 1|$.

r_1 EVENT-JOIN r_2

$= \{x_3 | x_3(R_1') = x_1(R_1') \wedge$

$$x_3(R_2') = x_2(R_2') \wedge$$

$$x_3(T_3) = x_1(T_1) \cap x_2(T_2)$$

$$\text{for } x_1 \in r_1 \;\&\; x_2 \in r_2,$$

$$\vee\; x_3(R_i') = x_i(R_i') \wedge$$

$$x_3(R_j') = \{\varnothing, \ldots, \varnothing\} \wedge$$

$$x_3(T_3) = \max\{|I| \mid I \subseteq x_i(T_i)\} \wedge$$

$$\text{there does not exist } x_j \text{ such that } x_j(S_j) = x_3(S_1) \;\&\; x_j(T_j) \cap x_3(T_3)$$

$$\text{for } i = 1, j = 2 \text{ or } i = 2, j = 1$$

}

4. FACTORS AFFECTING QUERY OPTIMIZATION

There are several important factors that distinguish the processing environment of temporal databases from conventional ones. We provide a brief introduction into several of them, and go into more detail over the selectivity estimation problem.

4.1. Data Organization

Temporal data may be organized in several ways. The first is a static organization, which is relevant for many scientific and statistical analysis. A second organization is to have data sorted according to a specified key order, reflecting the most common queries on the database. One possibility is to have data sorted by the key combination of surrogate and time start (S, T_S). A third organization is to take advantage of the clustering on T_S that results from append-only databases. Lastly, in a dynamic database, data may be left unsorted-- a query optimizer has to determine if it is worthwhile to specifically sort the data before processing a given query, or if it is better to use an unordered strategy. It is also possible that data is organized by a combination of the above methods, in the event that the database is partitioned into several segments, e.g., into an append-only historical store and a dynamic current time-window store.

4.2. Specialized Indexing Methods

Conventional indexing techniques may not be satisfactory performance-wise for temporal data retrieval. Several papers have been published in this field, e.g., [Lum et al 84, Rotem & Segev 87, Gunadhi & Segev 88, Kolovson & Stonebraker 89]. If appropriate indexing structures are developed, query response times may improve substantially. Research in this area

has focused on single relation operations, but there is the potential for performance gains if multirelational indexing is pursued.

4.3. Metadata

The maintenance and availability of statistical information about the temporal relation is a critical aspect of query processing. One important metadata is the lifespan of the relation, i.e. the time of the first event, and the current time or end of the last event. Where the database is segmented into more than one tier, there must be additional information on the current time-window. Moreover, statistical metadata may be required for such information as the the rate of arrival of new surrogate instances, departure of current instances and probability density functions for temporal attributes. Statistical data may be updated by random sampling for very large databases, or by a compile time scan.

4.4. Architecture of Query Processor

The final issue is the use, if any, of a conventional query processor for the processing of temporal queries. An implementation such as that of [Snodgrass & Ahn 87] is based on the construction of a temporal database on top of a conventional one. Minimal modification of the underlying processor is likely to cause inefficiencies in the processing of many temporal operators.

4.5. Estimation of Selectivities

Accurate cost estimation of relational operations is a crucial component of query optimization. A substantial amount of literature exists on selectivity estimation, among them by [Yao 77, Selinger et al 79, Christodoulakis 83, Piatetsky-Shapiro & Connell 84, Lynch 88, Mularikrishna & Dewitt 88, Ahad et al 89]. However, estimation techniques for snapshot relations cannot be readily applied to temporal relations. First of all, each relation consists of time-ordered histories of the modeled surrogates instances. Secondly, histories of surrogate instances may begin and end at different points in time. Third, some histories may be disjoint, i.e. there are intervals within it for which no data exists. Fourth, the temporal attributes themselves may also be time-dependent in behavior. Clearly, without modeling some or all of these properties explicitly, simple extension of existing methods will yield inaccurate results. Further, there are many operations, mainly joins, that cannot be estimated without explicit modeling, e.g. event-join and intersection join results. We will discuss the basic characteristics, desired measures and modeling approaches that can be taken.

4.6. Basic Characteristics of Temporal Relations

The following are the basic characteristics that have to be considered in modeling a relation with one temporal attribute.

Arrival of surrogate instances. Arrival of a new surrogate instance adds a new Time Sequence (TS) to the relation. Surrogates instances arrive according to some probability distribution; for example, a company may hire 120 new employees a year, at a uniform rate of 10 a month.

Departure and re-entry of surrogate instances. After arriving, a surrogate may remain for the duration of the relation's lifespan, leave permanently at some point, or leave and re-enter later. All these may be modeled by a single stochastic process, or perhaps by separate processes. If the surrogate instance is allowed to return, we assume that no new *TS* will be generated, instead the old *TS* of the surrogate instance is extended, but with a resulting discontinuity in its lifespan. As an example, from the EMP_MGR relation of Table 1, we can infer that employee E1 "arrives" at time 1, "leaves" at time 6, then "re-joins" at time 9; subsequently, a discontinuity is created in the lifespan of its *TS*, between tuples 1 and 2.

Arrival of tuples for a *TS*. The arrival process of tuples for a given *TS* follows some probability distribution representing the behavior of that surrogate. Further, each surrogate instance may have its own tuple arrival process, or may share an identical distribution with the other instances.

Distribution of temporal attribute values. We assume that each new tuple marks a change in the value of the temporal attribute of a particular surrogate; thus two consecutive tuples of a given *TS* must have different temporal attribute values, except when a discontinuity exists between two consecutive tuples. Attribute values may be time-dependent, in which case they can either be dependent on the event time itself, e.g. salaries paid based on seniority, or dependent on the value in one or more prior period(s), e.g. the value of a fixed deposit.

4.7. Multi-Attribute Temporal Modeling

A more complex scheme for a temporal relation is one involving multiple temporal attributes. We have to consider the interdependence amongst attributes in terms of both the timing of events and value changes in temporal attributes. In general, it would not be desirable to maintain relations where the temporal attributes are not synchronous [Navathe & Ahmed 86]. If such relations are maintained, then each new tuple indicates that one or more attributes have changed values. On the other hand, If the attributes are synchronous, we can model them as if they form a single attribute temporal attribute. In this case, the preceding discussions on modeling and measurement parameters directly apply.

4.8. Examples of Unary and Binary Estimates Needed

We will outline the main types of estimates needed for query processing purposes by using examples. For unary operations: (1) "How many employees were in the company between time 1 and 12?" (2) "Get all the manager records for E#1 between time 2 and 10." (3) "Find all commission records between time 4 and 10." and (4) "How many tuples in MANAGER have MGR = TOM between time 1 and 12?" For the case of binary estimates, they pertain to join sizes, i.e. the number of intersecting tuples for intersection type joins, the number of outerjoin tuples for an event-join, and the result of a time union join.

4.9. One-Attribute Model and Assumptions

We now introduce a model for the case of a single temporal attribute. The basic model consists of three independent probability distributions to describe the surrogate arrival process, tuple arrival process, and distribution of temporal attribute values. Several other parameters are added in order to increase the estimation power of the model.

Surrogate arrivals. Let $\{N^s_{r_i}(t)\}$, $t = 0, 1, 2, \cdots$ define the number of surrogate instance that arrive in period $(0, t]$ for relation r_i. We model $\{N^s_{r_i}(t)\}$ as a *Poisson* process with arrival rate λ^s_i.

Tuple arrivals. Let $\{N^c_{r_i}(t)\}$, $t = 0, 1, 2, \cdots$ be the number of tuple arrivals in period $(0, t]$ for an arbitrary surrogate instance in relation r_i. We model this counting process as a Poisson process with rate λ^c_i. The tuple arrival process for each surrogate instance is independent and identically distributed (i.i.d.).

Distribution of temporal attribute values. We model the temporal attribute values at different *change* time points during the surrogate instance's lifespan by an i.i.d. sequence of uniform random variables over the temporal attribute domain. Although it is incorrect to assume that for a given surrogate instance, two successive changes can yield the same temporal attribute value, the impact on estimation should not be significant if the relations and domain sizes are large. This approach is taken to simplify estimation, since time-dependent characterization requires knowledge of the actual behavior of the temporal attribute, which varies widely in reality.

Life-span of each TS. There are two ways in which to model the length of a surrogate instance's lifespan, which we denote as $LS^s_{r_i}$. The first method is to assume that it is as long as the lifespan of the relation itself, LS_{r_i}. The second way is to assume that it follows some probability distribution with mean $\overline{LS}^s_{r_i}$, and that the distribution for each instance in the surrogate domain is an i.i.d. random variable.

Treatment of Null Values. The null values in this model will be handled by using a parameter, called the existence density: $\delta_i^E = \frac{number\ of\ data\ points}{number\ of\ time\ points}$. Therefore $1 - \delta_i^E$ gives us the proportion of changes within a time sequence or relation that will generate nulls. Implicit is the assumption of uniformity in the distribution of nulls along surrogates' lifespans.

5. IMPLEMENTATION AND OPTIMIZATION OF TE-JOIN

We evaluate strategies for implementing the temporal equijoin and their associated costs. As an example, we will use the TE-join previously described in section 3 between the EMP_DEP and DEP relations on D#. Table 6 shows some statistics about the two relations. We make the following assumptions: (1) The values of D# is uniformly distributed throughout both relations; (2) Neither relation is sorted or clustered, and join processing is carried out by the nested-loop algorithm with DEP as the outer relation; (3) Each disk block holds 50 tuples of either relation; (4) The result relation, RESULT has 120,000 tuples or 2,400 pages; (5) The buffer size in main memory is $BUF = 20$ pages; and (6) No pipelining is used, which means that the temporary results ($TEMP_i$) are written to disk. The cost C_j of step j is measured in the number of disk I/O's.

Statistic	EMP_DEP	DEP
Relation Size (tuples), $\|r_i\|$	100,000	2000
Relation Size (pages), B_{r_i}	2,000	40
Number of Unique D#, $\|r_i(D\#)\|$	40	40
Number of Unique E#, $\|r_i(E\#)\|$	5,000	n.a.

Table 6. Statistics for Two Relations

We consider three approaches to the problem. The first illustrates a naive strategy, which would be the case if a temporal interface were to be built on top of a conventional system. The second strategy employs a standard theta-join operator where the time stamps are treated as ordinary attributes. In this case, a change is needed in the query processor to replace standard concatenation of tuples by its temporal equivalent. The third is an approach specifically designed for the TE-join, and requires a major change to the optimizer.

5.1. Naive Approach

In this strategy, the handling of the time attributes is ignored at the level of the conventional DBMS. Thus a simple equijoin on the non-time joining domains is executed, and the result is retrieved by a special temporal processor which carries out the restrictions over time

attributes, creates the new time stamps for qualifying tuples, and projects the final result. In other words, the logical steps carried out are as follows:

Step 1. $TEMP_1 \leftarrow EMP\_DEP\,[D\# = D\#]DEP$

Step 2. $TEMP_2 \leftarrow \sigma_{((EMP\_DEP.T_S \le DEP.T_E) \wedge (EMP\_DEP.T_E \ge DEP.T_S))}(TEMP_1)$

Step 3. $TEMP_3 \leftarrow \prod_{(TEMP_2 - T_{EMP\_DEP} - T_{DEP})}(TEMP_2)$ *CONCATENATE*

$$\{TEMP_3.T_S = max(EMP\_DEP.T_S, DEP.T_S),$$

$$TEMP_3.T_E = min(EMP\_DEP.T_E, DEP.T_E)\}$$

Step 4. $RESULT \leftarrow \prod_{(NAME,\ D\#,\ T_S,\ T_E)}(TEMP_3)$

We divide the operation into four steps for clarity of exposition. The *CONCATENATE* operator in Step 3 is introduced to allow the appending of attributes not directly created by a join or cross product. Note also that in Step 3 we distinguish between the similarly named time-stamps in the temporary relation by qualifying them on their original relations. The I/O cost is computed in the following manner. For step 1,

$$C_1 = B_{DEP} + \left\lceil \frac{B_{DEP}}{BUF} \times B_{EMP\_DEP} \right\rceil + B_{TEMP_1},$$

which represents the cost of nested-loop execution plus the cost of writing the temporary result to disk. $TEMP_1$ is the result of a conventional equijoin, which means that a cross product on the time domains is carried out for qualifying tuples. Given our uniformity assumption,

$$C_1 = 20 + (2 \times 2{,}000) + \frac{|EMP\_DEP| \times 50}{50} = 104{,}020.$$

We assume that steps 2 to 4 are executed in a single scan, i.e. $C_{2\text{-}4} = P_{TEMP_1} + P_{RESULT}$ = 102,400. The total cost of this approach is therefore 206,420 disk I/O's.

5.2. Theta-Join Strategy

In this strategy, we convert the intersection predicate on time into a conjunction of inequality predicates on the time attributes, and treat them as "ordinary" predicates. The query is then processed as a conventional theta-join. Since the creation and concatenation of the new time attributes is unique to temporal data, these operations will still be carried out separately by a temporal processor. The strategy is made up of the following steps:

Step 1. $TEMP_1 \leftarrow EMP\_DEP\,[D\# = D\#]DEP$ *WHERE*

$$EMP\_DEP.T_S \leq DEP.T_E \wedge$$

$$EMP\_DEP.T_E \geq DEP.T_S$$

Step 2. $TEMP_2 \leftarrow \prod_{(TEMP_1 - T_{EMP\_DEP} - T_{DEP})}(TEMP_1)$ *CONCATENATE*

$$\{TEMP_2.T_S = max(EMP\_DEP.T_S, DEP.T_S),$$

$$TEMP_2.T_E = min(EMP\_DEP.T_E, DEP.T_E)\}$$

Step 3. $RESULT \leftarrow \prod_{(NAME,\ D\#,\ T_S,\ T_E)}(TEMP_2)$

Steps 2 and 3 are identical to steps 3 and 4 of the previous strategy. The total cost is the sum of the cost of reading in the two relations by the nested-loop method, the cost of writing $TEMP_1$ and the cost of reading in $TEMP_1$ and writing $RESULT$. Since $TEMP_1$ and $RESULT$ are of the same size, the total cost comes to 4,020 + 3 × 2,400 = 11,220. This is considerably lower than the previous method. In this case we were able to transform a temporal operation to an equivalent conventional one (from the point of view of optimization); we are constrained, however, in this approach to the non-temporal nature of a traditional optimizer. Also, some temporal operators cannot be translated into equivalent relational operators, e.g. the event-join operator.

5.3. Directly Implementing TE-JOIN

The TE-join operator can be implemented independently. There are two primary issues: (1) The manner in which comparison between the tuples is carried out and (2) How concatenation of the new time attributes is achieved. The previous approaches required time-stamp comparisons to be evaluated twice, but we can create the new time stamps for the result tuple, i.e. find $T_S^* = max(EMP\_DEP.T_S, DEP.T_S)$ and $T_E^* = min(EMP\_DEP.T_E, DEP.T_E)$, then concatenate them iff they are satisfied by the predicate $T_S^* \leq T_E^*$. This test substitutes for the intersection predicate on the two relations' time subschemes.

In algebraic terms, we execute the query as follows:

$$\prod_{(R_1' \cup R_2' \cup (T_S^*, T_E^*))}\left[\sigma_{\wedge_j r_1.A_{1j} = r_2.A_{2j} \wedge T_S^* \leq T_E^*}\right]((r_1 \times r_2)\ CONCATENATE\ (T_S^*, T_T^*))$$

The following procedure executes it.

```
for each x1 ∈ r1 {
    for each x2 ∈ r2 {
        find TS* and TE*
        for p ∈ PR' ∧ PT
            if not p, do the next x2
        else output tuple on scheme (R1' ∪ R2 ∪ (TS*, TE*))
    }
}
```

The total cost is merely the cost of reading in the relations for the nested-loop method and the cost of writing the output. This comes to 6,420 pages, which is cheaper than the cost of the second strategy. Bear in mind that the sizes of the example relations are relatively small, and the savings would be even more significant for joins involving very large relations.

6. SUMMARY AND FUTURE RESEARCH

We have introduced and defined four classes of temporal joins: Theta, intersection, union and event joins. We believe that these joins can be used for a large number of join-type queries which have been introduced but not formally defined or identified by others. Moreover, we have developed a framework within which we can evaluate techniques that can optimize the execution of queries involving such joins. We show by example that there are inherent differences between using conventional query processors and developing specialized procedures and algebra to solve these queries. We must remember that the time attributes in a tuple-versioning model must always be treated differently than other attributes, although in many algebraic operations, they may be qualified with the same type of predicates as non-time attributes. Further, we showed that the selectivity estimation problem is of even greater importance for temporal relations, and that it cannot be modeled in the same way as conventional relations.

Current and future research address the following issues:

- Developing selectivity estimates based on the model presented, and to expand the scope and sophistication of the model itself. There is also a tradeoff between accuracy of estimates, and the expense of maintaining the necessary statistics and deriving the estimates.
- Investigating the optimization of each class of join. For the temporal equijoin, we are looking at algorithms that exploit data ordering and specialized indexing. Further, the event-join operator is likely to be a commonly used operator, and yet it has no equivalence in the "snapshot" database context. Comprehensive tests of the efficiency of alternatives algorithms are necessary.

- Extending the investigation of temporal operators to those involving temporal ordering and aggregation.
- Continuing our study into the design of efficient data structures, in order to improve the data retrieval capability of a temporal DBMS.

REFERENCES

[Adiba & Quang 86] Adiba, M, Quang, N.B., Historical Multi-Media Databases, *Proceedings of the International Conference on Very Large Databases*, Aug. 1986, pp. 63-70.

[Ahad et al 89] Ahad, R., Rao, K.V.B., McLeod, D., On Estimating the Cardinality of the Projection of a Database Relation, *ACM Transactions on Database Systems*, 14, 1, Mar. 1989, pp. 28-40.

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

[Christodoulakis 83] Christodoulakis, S., Estimating Record Selectivities, *Information Systems*, 8, 2, 1983, pp. 105-115.

[Clifford & Croker 87] Clifford, J., Croker, A., The Historical Relational Data Model (HRDM) and Algebra Based on Lifespans, *Proceedings of the International Conference on Data Engineering*, Feb. 1987, pp. 528-537.

[Clifford & Tansel 85] Clifford, J., Tansel, A., On an Algebra for Historical Relational Databases: Two Views, *Proceedings of ACM SIGMOD International Conference on Management of Data*, May 1985, pp. 247-265.

[Kolovson & Stonebraker 89] Kolovson, C., Stonebraker, M., Indexing Techniques for Historical Databases, *Proceedings of the International Conference on Data Engineering*, Feb. 1989, pp. 127-139,

[Gunadhi & Segev 88] Gunadhi, H., Segev, A., Physical Design of Temporal Databases, Lawrence Berkeley Lab Technical Report LBL-24578, January 1988.

[Lynch 88] Lynch, C.A., Selectivity Estimation and Query Optimization in Large Databases with Highly Skewed Distribution of Column Values, *Proceedings of the International Conference on Very Large Databases*, Aug. 1988, pp. 240-251.

[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 ACM SIGMOD International Conference on Management of Data*, Jun.1984, pp. 115-130.

[Mulakrishna & DeWitt 88] Mulakrishna, M., DeWitt, D.J., Equi-Depth Histograms for Estimating Selectivity Factors for Multi-Dimensional Queries, *Proceedings of ACM SIGMOD International Conference on Management of Data*, May 1988, pp. 28-36.

[Klopproge & Lockemann 83] Klopproge, M.R., Lockemann, P.C., Modeling Information Preserving Databases: Consequences of the Concepts of Time, *Proceedings of the International Conference on Very Large Databases*, Aug. 1983, pp. 399-416.

[Navathe & Ahmed 86] Navathe, S., Ahmed, R., A Temporal Relational Model and a Query Language, UF-CIS Technical Report TR-85-16, Univ of Florida, April 1986.

[Piatetsky-Shapiro & Connell 84] Piatetsky-Shapiro, G., Connell, C., Accurate Estimation of the Number of Tuples Satisfying a Condition, *Proceedings of ACM SIGMOD International Conference on Management of Data*, May 1984, pp. 256-276.

[Rosenthal & Reiner 84] Rosenthal, A., Reiner, D., Extending the Algebraic Framework of Query Processing to Handle Outerjoins *Proceedings of the International Conference on Very Large Databases*, Aug. 1984, pp. 334-343.

[Rotem & Segev 87] Rotem, D., Segev, A., Physical Organization of Temporal Data, *Proceedings of the International Conference on Data Engineering*, Feb. 1987, pp. 547-553.

[Segev & Gunadhi 89] Segev, A., Gunadhi, H., Event-Join Optimization in Temporal Relational Databases, *Proceedings of the International Conference on Very Large Databases*, Aug. 1989. pp. 205-215.

[Segev & Shoshani 87] Segev, A., Shoshani, A., Logical Modeling of Temporal Databases, *Proceedings of ACM SIGMOD International Conference on Management of Data*, May 1987, pp. 454-466.

[Segev & Shoshani 88a] Segev, A., and Shoshani, A., The Representation of a Temporal Data Model in the Relational Environment, *Lecture Notes in Computer Science*, Vol 339, M. Rafanelli, J.C. Klensin, and P. Svensson (eds.), Springer-Verlag, 1988, pp 39-61.

[Segev & Shoshani 88b] Segev, A., Shoshani, A., Functionality of Temporal Data Models and Physical Design Implementations, *IEEE Data Engineering*, 11, 4, Dec. 1988, pp. 38-45.

[Selinger et al 79] Selinger, P.G., Astrahan, M.M., Chamberlain, D.D., Lorie, R.A., Price, T.G., Access Path Selection in a Relational Database System, *Proceedings of ACM SIGMOD International Conference on Management of Data*, May 1979, pp.23-34.

[Shoshani & Kawagoe 86] Shoshani, A., Kawagoe, K., Temporal Data Management, *Proceedings of the International Conference on Very Large Databases*, Aug. 1986, pp. 79-88.

[Snodgrass 87] Snodgrass, R., The Temporal Query Language TQuel, *ACM Transactions on Database Systems*, Jun. 1987, pp. 247-298.

[Snodgrass & Ahn 85] Snodgrass, R., Ahn, I., A Taxonomy of Time in Databases, *Proceedings of ACM SIGMOD International Conference on Management of Data*, May 1985, pp. 236-246.

[Snodgrass & Ahn 87] Snodgrass, R., Ahn, I., Performance Analysis of Temporal Queries, TempIS Document No. 17, Department of Computer Science, University of North Carolina, August 1987.

[Yao 77] S.B. Yao, Approximating Block Accesses in Database Organizations, *Communications of the ACM*, 20, 4, Apr. 1977, pp. 260-261.