VLDB Journal, 2 (1):75-111 (1993) Fred J. Maryanski, Editor
©VLDB

Using Differential Techniques to Efficiently Support Transaction Time

Christian S. Jensen, Leo Mark, Nick Roussopoulos, and Timos Sellis

Received May 30, 1990; revised version received February 28, 1992; accepted June 25, 1992.

Abstract. We present an architecture for query processing in the relational model extended with transaction time. The architecture integrates standard query optimization and computation techniques with new differential computation techniques. Differential computation computes a query incrementally or decrementally from the cached and indexed results of previous computations. The use of differential computation techniques is essential in order to provide efficient processing of queries that access very large temporal relations. Alternative query plans are integrated into a state transition network, where the state space includes backlogs of base relations, cached results from previous computations, a cache index, and intermediate results; the transitions include standard relational algebra operators, operators for constructing differential files, operators for differential computation, and combined operators. A rule set is presented to prune away parts of state transition networks that are not promising, and dynamic programming techniques are used to identify the optimal plans from the remaining state transition networks. An extended logical access path serves as a "structuring" index on the cached results and contains, in addition, vital statistics for the query optimization process (including statistics about base relations, backlogs, and queries—previously computed and cached, previously computed, or just previously estimated).

Key Words. Temporal databases, transaction time, efficient query processing, incremental and decremental computation.

1. Introduction

The relational model presented by E. F. Codd twenty years ago (Codd, 1970, 1979) has gained immense popularity and is regarded today as a defacto standard for business applications. A main reason for the success is the generality of the model; it makes

Christian Jensen, Ph.D., is Assistant Professor, Department of Mathematics and Computer Science, Aalborg University, Fr. Bajers Vej 7, DK-9220, Øst, Denmark. Leo Mark, Ph.D., is Associate Professor, College of Computing, Georgia Tech, Atlanta, GA 30332, USA. Nick Roussopoulos, Ph.D., is Full Professor; and Timos Sellis, Ph.D., is Associate Professor, Department of Computer Science, University of Maryland, College Park, MD 20742, USA.

very few assumptions about specific application areas. This, however, has its drawbacks because the model does not provide detailed and customized support for some application areas. Extensions that make the relational model more suitable for the application areas have been a topic of interest in the database research community ever since the relational model was presented.

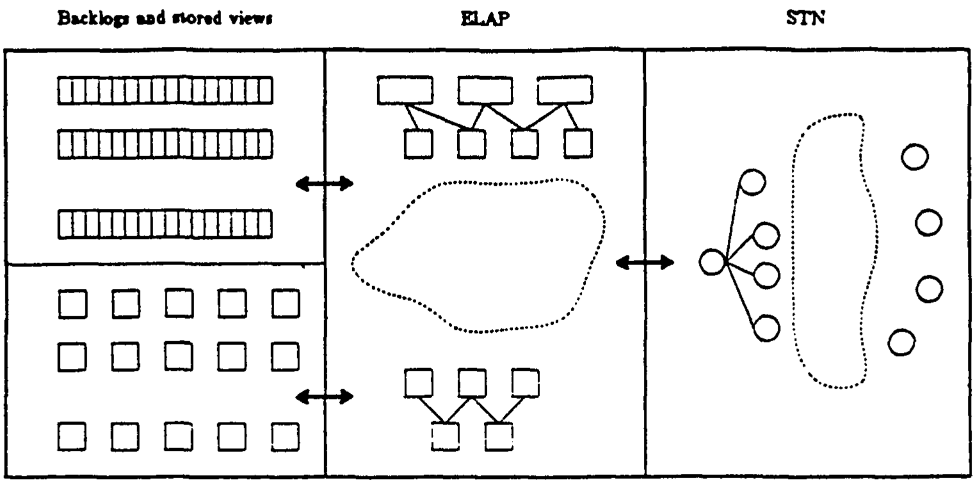
This article presents an implementation model, IM/T (Implementation Model/Time), for an extension of the relational model supporting transaction time, DM/T (Data Model/Time) (Jensen et al., 1991, 1992). Data are never deleted once entered into a database in this model; it is possible to see the database from any time in the past, and it is possible to analyze the change history. Many applications will benefit from efficient transaction time support. In the literature, engineering, econometrics, banking, inventory control, medical records, and airline reservations have been mentioned as candidates (McKenzie and Snodgrass, 1991).

Traditional implementation models cannot cope efficiently with huge, ever growing quantities of historical data. The predominant approach taken to solve this problem has been partitioned storage, where data of individual relations are partitioned, and a storage hierarchy is maintained that favors efficient support of queries solely accessing recent data (Lum et al., 1984; Salzberg and Lomet, 1989). While still allowing for partitioned storage, the data organization of IM/T allows efficient access to frequently accessed states of individual relations, recent or old, thus providing efficient support of any state.

IM/T exploits caching of query results. Caching is the idea of storing results, on secondary memory, of previous computations and subsequently using them to avoid redoing expensive computations (Roussopoulos, 1982*b*, Sellis, 1988*a*). Caching trades replication of data for speed of retrieval. It is potentially a very powerful technique, but a number of issues must be dealt with intelligently in order to gain the full benefits. Let us mention the most important ones, some of which are addressed in this article while others are still issues for future research.

First, there is the question of how to cache results. In IM/T, query results can be stored as actual data or as pointers to base data, possibly via several levels of indirection. Pointer cache storage gives a fixed, small tuple size and makes results very compact thus allowing for efficient use of main memory (Roussopoulos, 1991). For transaction-time databases, however, one base data page must be read for each pointer in extreme cases. Data cache storage solves this potential problem because it allows for control of locality of reference. Additionally, it allows for reduction of references to slower storage areas. While the architecture allows for both data and pointer caching, a detailed study of the relative merits of the two is still warranted.

Figure 1. Three IM/T stores: base data, derived data, ELAP



IM/T has three stores, one for base data, one for derived data, and one for the ELAP, containing statistics and representing the structure of base and derived data. During query optimization, plans using the stored data are enumerated in STNs.

Second, the utility of caching can be improved by means of cache indexing. IM/T extends the logical access path (Roussopoulos, 1982*b*) into an Extended Logical Access Path (ELAP), which allows for efficient identification of all potentially useful results during query processing. It is a persistent query graph with nodes for all cached, computed, or just estimated results. While the algorithms for maintaining and using the ELAP have not been developed, it has been demonstrated that an appropriate extension of the algorithms for the logical access path is fairly straightforward, i.e., the rule-access path (Sellis et al., 1990).

Third, to gain the full benefits, caching should be used in conjunction with differential computation techniques (Roussopoulos, 1991). The application of such techniques prolongs the usefulness of cached results because slightly outdated results need not be discarded and recomputed, but can instead be efficiently incremented or decremented to answer a query. IM/T generalizes incremental computation to differential computation using both incremental and decremental techniques, and it unifies differential computation and traditional recomputation. Differential computation is the focus of this article, and it is treated in great detail.

Fourth, only potentially beneficial results should be cached. If the cache is full, appropriate replacement strategies must be used. IM/T has a cache management component that supports selective caching and cache replacement. The purpose for selective caching is that neither caching of all results (and differential computation) or no caching at all (and recomputation) is superior to the other in every given situation. Caching is attractive in environments characterized by many queries, few updates, very large underlying base relations, and comparably small results. Methods of adapting the numerous contributions on cache management into appropriate strategies for selective caching and cache replacement in this context are discussed elsewhere (Hanson, 1987; Sellis, 1987, 1988*a*; Jhingran, 1988; Jhingran and Stonebraker, 1989).

Fifth, the fact that cached results become outdated must be addressed. Any possible update strategy ranging from "eager" (i.e., when relevant base data are entered), over threshold-triggered, to "lazy" (i.e., when the result is requested) is possible (Roussopoulos and Kang, 1986; Hanson, 1987). The details of cache updating are not part of this article.

In a temporal setting the maintenance of stored results is likely to be more feasible than in a snapshot setting. The reasons are that relations are large because previous states are retained and essential additional semantics for the process of selective caching is available. For example, fixed views are primary candidates for caching because they never become outdated, and the future outdatedness of time-dependent views issued against past states can be estimated at the time of computation.

Query-plan generation in IM/T uses the concept of state transition network (STN; Lafortune and Wong, 1986). Query-plan selection uses dynamic programming (see Figure 1). We present a set of rules for pruning the STNs generated, the idea being to avoid generating inferior paths, thus saving both space and time during cost estimation. During query-plan generation and selection we use results from the cache, and we use both recomputation and differential computation versions of the operators of the query language of DM/T as possible transitions in STNs. Apart from defining the operators, we discuss how to efficiently implement the differential versions. In addition, combined operators are introduced to minimize the need for storage of intermediate results during query computation.

Efficient query processing is a central theme in database research, and consequently the work of this article is related to a number of previous efforts.

The transaction-time extension of this article was designed to be transparent to the naive user of the standard relational model. To our knowledge, none of the other temporal extensions of the relational model shares this characteristic (Bubenko, 1977; Bolour et al., 1982; Snodgrass and Ahn, 1985; Snodgrass, 1987; Stam and Snodgrass, 1988; McKenzie and Snodgrass, 1991).

IM/T allows for partitioned storage and supports both reverse and forward chaining. Related efforts can be found (Dadam et al., 1984; Lum et al., 1984; Ahn, 1986; Snodgrass and Ahn, 1988; Kolovson and Stonebraker, 1989; Salzberg and Lomet, 1989). Grid files have been suggested as a means of implementation of temporal data (Shoshani and Kawagoe, 1986), but they seem inappropriate because surrogates, for which no natural ordering exists, would be one dimension and time the other. In addition, indexing of other attributes is not allowed, which again is unsatisfactory. The subject of Rotem and Segev (1987) is multi-dimensional file partition for static files with time as one of multiple dimensions.

Some research (Gunadhi and Segev, 1989; Gunadhi et al., 1989; Segev and Gunadhi, 1989) concentrates on different kinds of temporal joins (time-union, time-intersection, and event-joins) and temporal-selectivity estimation. This research, while interesting, is not addressed here.

The focus of the work presented by McKenzie (1988) is the data model for a temporal database, and it is closely related to our work. It formally defines incremental algebra operators, resembling those of our state-transition space. In addition, it surveys applications of incremental techniques in the relational model, and discusses ways to combine previous efforts into an implementation supporting both transaction time and valid (logical) time. Our work concentrates only on implementation and on transaction time. We present a detailed design of an implementation model and concentrate on query optimization and processing.

IM/T exploits caching of views and the literature contains many contributions to the understanding of its many aspects. Aspects of materialized views relevant to distributed processing are presented in Segev and Fang (1989, 1990). The performance of three techniques (lazy incremental computation, eager incremental computation, and recomputation) has been compared by Hanson (1987), who demonstrated that none of the techniques were superior to the others in all cases. Caching of query results has been addressed to support query language procedures (programs, rules) efficiently stored in relational fields (Jhingran, 1988; Sellis, 1987, 1988*a*). Techniques aimed at reducing the cost of maintaining materialized views have been recently reported by Blakely et al. (1986, 1989) who attempt to detect base data updates that do not affect a view, and to detect when a view can be correctly updated using only the data already present. IM/T generalizes and unifies traditional recomputation and incremental computation so that a single query can be processed using re-computation, incremental computation, and decremental computation. Traditional systems, e.g., Ingres (Wong and Youseffi, 1976) and System R (Selinger et al., 1979) use recomputation. Kinsley and Driscoll (1979, 1984) have described how to extend the RAQUEL II database management system to support dynamic derived relations using eager incremental update. In ADMS (±), a database management system implementing the standard relational model, incremental computation of views stored as pointer structures is used (Roussopoulos, 1982*a*, 1987, 1991). Our work has some resemblance to Postgres, where previous history is also retained. The temporal support, however, never was the focus, and time stamps and backlog queries are not supported as in IM/T. Postgres exploits caching, but since indexing, differential cache maintenance, and query execution are missing, the full potential of caching is not achieved (Rowe and Stonebraker, 1987).

For previous work on query optimization, and further references, see Smith and Chang (1975), Selinger et al. (1979), Jarke and Koch (1984), and Sellis and Shapiro (1985).

State transition networks have, to our knowledge, never been applied in a temporal setting or in settings involving caching. Lafortune and Wong (1986) used STNs as a framework for query optimization in a distributed environment. Hong and Wong (1989) applied STNs to multiple query optimization.

The structure of the remaining part of this article is as follows: Section 2 serves as a specification of the functionality to be supported by IM/T. The concept of transaction time, data structures, and the query language of DM/T are presented. The remaining sections are devoted to IM/T and the efficient processing of DM/T queries. Section 3 describes the three stores of IM/T—base data, cache, and ELAP. In Section 4, STNs are used for enumerating alternative query plans and dynamic programming is used to collect costs of entire plans from costs of single transitions. The concrete state and transition spaces, incorporating the use of cached results, differential computation, standard query computation techniques, and support for combined operators, are introduced. Also discussed is the use of ELAP to find promising results from the cache, considered when STNs are generated. In Section 5 we first present the cases to consider when implementing operators and then discuss the three types of operators: Recomputation operators, operators that construct differential files, and differential operators. Section 6 presents rules for reducing the sizes of the generated STNs. Section 7 concludes this article.

2. Transaction Time in the Relational Model, DM/T

In this section we briefly introduce the transaction time extension of the basic relational model (Codd, 1970, 1979; Jensen et al., 1991). Our purpose is to identify the kinds of queries that should be supported by IM/T. The properties of the time concept offered by DM/T are outlined in Figure 2 and are discussed below.

Figure 2. Characterization of the time concept offered by DM/T.

$$\left\{\begin{array}{c}\boxed{\text{transaction}}\\ \text{logical}\end{array}\right\} \times \left\{\begin{array}{c}\text{regular}\\ \boxed{\text{irregular}}\end{array}\right\} \times \left\{\begin{array}{c}\text{discrete}\\ \boxed{\text{stepwise cont.}}\end{array}\right\} \times \left\{\begin{array}{c}\boxed{\text{true}}\\ \text{arbitrary}\end{array}\right\} \times \left\{\begin{array}{c}\boxed{\text{automatic}}\\ \text{manual}\end{array}\right\}$$

Two orthogonal time dimensions have been studied in temporal databases (Snodgrass and Ahn, 1985). Logical time models time in the part of reality modeled by a database. Transaction time models time in the part of the reality that surrounds the database, the input subsystem. While logical time is application-dependent, transaction time depends only on the database management system, and is inherently application-independent.

First, DM/T supports transaction time as opposed to logical time. Second, a domain is regular if the distances between consecutive values of the active domain are identical. Otherwise the domain is irregular. DM/T supports an irregular time domain. Third, a time domain can be discrete or stepwise continuous. Tuples with discrete timestamps are only valid at the exact times of their timestamps. In contrast, tuples have an interval of validity in a stepwise continuous domain. The DM/T time domain has this property (also termed *stability*) because the values of a relation remain the same until the relation is changed by a new transaction. Fourth, DM/T supports true time as opposed to arbitrary time. True time reflects the actual time of the input subsystem while an arbitrary time domain only needs to have a metric and a total order defined on it; the set of natural numbers is a possible arbitrary time domain. Fifth, DM/T has automatic time-stamping, which is the natural choice for transaction time. Manual, user-supplied timestamp values are natural for logical time. We have chosen tuple stamping as opposed to attribute value stamping. The major reason has been to provide a first normal-form model which is a simple and yet powerful extension of the standard relational model.

In order to record detailed temporal data and still be able to use the operators of the basic relational model, we have introduced the concept of a backlog relation. A *backlog*, B_R, for a relation, R, is a relation that contains the complete history of change requests to relation R (Roussopoulos and Kang, 1986). Backlog B_R contains three attributes in addition to those of R. Attribute *Id* is defined over a domain of logical, system generated unique identifiers, i.e., surrogates. The values of *Id* represent the individual tuples, termed *change requests*. The attribute *Op* is defined over the enumerated domain of operation types, and values of *Op* indicate

Figure 3. System-controlled insertions into a backlog.

Requested operation on R:	Effect on B_R:
insert R(tuple)	insert B_R(id, *Ins*, time, tuple)
delete R(key)	insert B_R(id, *Del*, time, tuple(key))
modify R(key, new value)	insert B_R(id, *Mod*, time, tuple(key,new value))

The function "tuple" returns the tuple identified by its argument.

whether an insertion (*Ins*), a deletion (*Del*) or a modification (Mod) is requested.[1] Finally, the attribute *Time* is defined over the domain of transaction timestamps, *TTIME*, as previously discussed. DM/T automatically generates and maintains a backlog for each base relation (i.e., user-defined relations and schema relations). Figure 3 shows the effect on backlogs resulting from operation requests on their corresponding relations.

As a consequence of the introduction of timestamps, a base relation is now a function of time. To retrieve a base relation it must first be *time sliced*. To define timeslice, assume that R has the attributes $A_1, A_2, \ldots, A_n$ and let $t \in [t_{init}; NOW]$ where t_{init} is the time when the database is initialized and *NOW* is a special variable with the current time as its value. Now, R at time t is defined as follows:

$$\begin{aligned} R(t) \;=\; & \{x \mid \exists s(B_R(s) \wedge x[1] = s[1] \wedge x[2] = s[2] \wedge \ldots \wedge x[n] = s[n] \wedge \\ & s[Time] \leq t \wedge (s[Op] = Mod \vee s[Op] = Ins) \wedge \\ & (\neg \exists u(B_R(u) \wedge s[R.Id] = u[R.Id] \wedge s[Time] < u[Time] \leq t)))\} \end{aligned}$$

When the database is initialized, it has no history and every relation is empty. If R is parameterized with an expression that evaluates to a time value, then the result is the state of R as it was at that point in time. It has no meaning to use a time before the database was initialized and after the present time. If R is used without any parameters this indicates the current R, i.e., $R \overset{def}{=} R(NOW)$. Time sliced relations have an *implicit* time stamp attribute, not shown unless explicitly projected. Note that these features help provide transparency to the naive user.

1. At a lower level, modifications are modeled by a deletion followed by an insertion, each with the same timestamp value.

If the expression E of a time-sliced relation $R(E)$ contains the variable *NOW*, then R is *time dependent*. Otherwise, it is *fixed*. While fixed-time slices of relations never get outdated, time-dependent time slices do, and they are consequently updated by the DBMS before retrievals.

A view is time-dependent if it is derived from at least one of the time-dependent relations and views. Otherwise it is fixed. Traditional views are ultimately derived directly and solely from time-sliced base relations. If a view ultimately is derived directly (i.e., not via a time-sliced base relation) from at least one backlog, then we term it a *backlog view*. Backlog views are time sliced as are base relations and views. Backlog view time slices involving *NOW* are time-dependent, and, as above, so are backlog views derived from views involving *NOW*. We define:

$$R(t_x) \overset{\mathrm{def}}{=} \sigma_{Time \leq t_x} B_R$$

$$B_R \overset{\mathrm{def}}{=} B_R(\mathit{NOW}).$$

By introducing the time slice operator it is possible to use the standard relational algebra as the query language. The query language of DM/T was presented in Jensen et al. (1991), and in Jensen and Mark (1992) it was extended to support analysis of change history. In this article we only consider time-slice, selection, projection, and equi-join. We adopt a set of precedence rules to simplify the appearance of query expressions. Time-slice has highest precedence, and is followed by projection and selection with the same precedence, which, in turn, are followed by binary operators, all with the same precedence. Parentheses are used to control precedence in the standard way, and evaluation is from left to right.

3. Structures of the Implementation Model, IM/T

In the previous section we described the data model, DM/T. The subject of this and the remaining sections is the implementation model, IM/T, which supports the data structures and operators of DM/T. We present the three different stores of IM/T: the store containing backlogs and indices; the cache containing views; and the ELAP which contains information about queries, and is an index to the cache.

3.1 Storage of Backlogs

Backlogs assume the role of base relations and are always stored. They are stored like traditional base relations with the possibilities of traditional indexing. Throughout this article we assume that tuples of a backlog are sorted according to the values of their transaction timestamp attribute. Also, mainly for simplicity, we assume that backlog tuples actually contain all the data of their attributes—compression techniques (Bassiouni, 1985) may be applied to the backlogs. To further cope with the ever-growing bulk of historical data, partitioned storage techniques may be introduced (Dadam et al., 1984; Lum et al., 1984; Ahn, 1986; Christodoulakis, 1987; Snodgrass and Ahn, 1988; Kolovson and Stonebraker, 1989; Salzberg and Lomet, 1989).

Finally, realizing that even WORM storage is limited and that some historical data might not be needed by any user, we have offered advanced facilities for pruning historical data elsewhere (Jensen and Mark, 1990).

3.2 Pointer and Data Cache of IM/T

The cache of IM/T is a collection of query results stored as either pointers or data. A part of secondary memory is allocated for the cache. Each entry of the cache is of the form *(rid, result)* where *rid* uniquely identifies an entry and *result* is of the format

$$\textit{result} \leftarrow \textit{array of ptr} \mid \textit{array of (ptr} \times \textit{ptr)} \mid \textit{relation}$$

Tuples of the same entry are stored consecutively and are sorted on *tid's* (pointers) or surrogate attribute values (data). Indices can exist on the tuples of results.

The ELAP, discussed in the next subsection, is a structuring index on the cache and is used to identify cache entries to be used in query processing. In the ELAP, a cache entry is represented by its *rid*, and therefore an index of *rid* entry results is desirable.

Differential files computed as intermediate results during query processing are not stored in the cache. It may, however, be useful to store statistics about such files. Such statistics may help estimate the cost of processing future differential files and help choose between different ways of processing a differential file. The design of data structures and algorithms that maintain the statistics, and the use of the statistics during query optimization are subjects of current research.

The cache contains the current states of all base relations, and they are updated readily. This makes the extended data model DM/T transparent to the naive user and enables IM/T to retrieve current data and check standard integrity constraints efficiently.

3.3 The Extended Logical Access Path of IM/T

The ELAP is a directed acyclic graph (DAG) (Roussopoulos, 1982*b*). Each node is associated with a set of equivalent query expressions, a list of statistics about each query expression, and an optional reference to a cached result. The edges are labeled by operators, and in the unary case an edge from node N_a to node N_b indicates that the operator constructs an expression associated with N_b from an expression associated with N_a. In the binary case, a pair of edges, possibly ordered, from nodes N_a and N'_a to node N_b indicates that the operator constructs an expression associated with N_b from expressions associated with N_a and N'_a. Here, we allow time slice, selection, projection, and join as labels of edges of the ELAP. In addition, we allow for combined operators in order to avoid the storage of intermediate results.

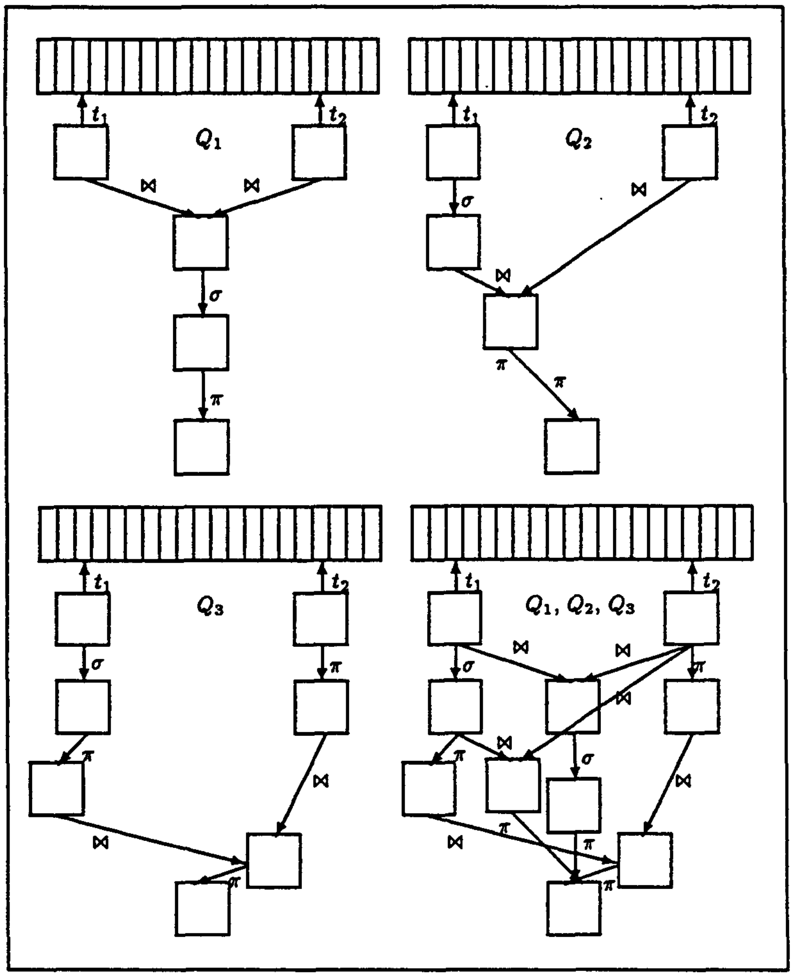
The ELAP integrates graphs of query expressions that have been computed or have been subject to estimation of statistics into a unifying structure by merging nodes representing common (sub-)expressions. It is important to observe that, while the expressions of a node all produce the same result, they may have different processing costs. The ELAP is a generalized AND/OR DAG where, at a single node, there is a choice ("OR") of one of several sets of "AND" edges (Rich, 1983; Mahanti and Bagchi, 1985), where "AND" edges correspond to binary operators. To illustrate, consider the following three equivalent query expressions defined on an employee relation *Emp* with attributes *Id* (employee id), *Sal* (salary), and *Dep* (department).

$$\begin{aligned}
Q_1 \quad & \pi_{Emp(t_1).Id,Emp(t_2).Sal}\big(\sigma_{Emp(t_1).Sal\geq 30}\big((Emp(t_1)\big) \\
& \qquad \bowtie_{Emp(t_1).Id=Emp(t_2).Id} \big(Emp(t_2)\big)\big)\big) \\
Q_2 \quad & \pi_{Emp(t_1).Id,Emp(t_2).Sal}\big(\sigma_{Sal\geq 30} \big(Emp(t_1)\big) \\
& \qquad \bowtie_{Emp(t_1).Id=Emp(t_2).Id} \big(Emp(t_2)\big)\big)\big) \\
Q_3 \quad & \pi_{Emp(t_1).Id,Emp(t_2).Sal}\big(\pi_{Id}\big(\sigma_{Sal\geq 30}\big(Emp(t_1)\big)\big)\big) \\
& \qquad \bowtie_{Emp(t_1).Id=Emp(t_2).Id} \big(\pi_{Id,Sal} \big(Emp(t_2)\big)\big)\big)\big)
\end{aligned}$$

Each query returns the *Id*'s and *Sal*'s at time t_2 of employees that were employed at both time t_1 and t_2 and that earned more than $30,000. at t_1. Yet, they are different expressions with different processing characteristics. The ELAP for these expressions is shown in Figure 4.

It follows that a cached result of a node could have been computed in several ways, and that it subsequently can be computed in several ways. A node tells from which expression a cached result was most recently computed. There is at most one cache entry per node.

Figure 4. Three equivalent query expressions.



The view corresponding to a node can be computed from several query language expressions. The figure represents three equivalent query expressions, first separate and then combined.

Nodes can belong to one of several categories, depending on the computational status of the labeling query expressions. The result of a query expression can be cached as data or pointers; the result of the query expressions can have been cached previously as data or pointers; it is possible that no result of the query expressions has ever been cached, but results might have been computed or just estimated;

finally, a node can denote a backlog.

Different types of statistics can be kept in each of the six types of nodes. Individual statistics should only be maintained if the cost of doing so is less than the benefits achieved from having them available during query optimization. Practical experiments are needed to determine when this is the case. Possible statistics include: cardinality of stored result; result stored as pointer or data; tuple size; which expression is cached; up-to-date status; how often used; usage; computation cost; when deleted; why deleted; and available indices.

4. Query Plan Generation and Selection

To efficiently compute a query, the system generates a state transition network (STN) where the initial state contains the uncomputed query, the backlog relations (in which terms it is defined) the cache, and the ELAP. A state transition occurs when the cost of a partial computation toward the total computation of the query is estimated. The new state is identical to the predecessor state except it is assumed that the cost-estimated computation has been performed. A final state is reached when the costs of all computations have been estimated. By following all paths from the initial to a final state and accumulating costs for each path, the total costs of computing the query in different ways are obtained, and we can choose the query plan with the lowest cost. The purpose of this section is to formalize and elaborate on the generation of query plans as just described.

4.1 State Transition Network

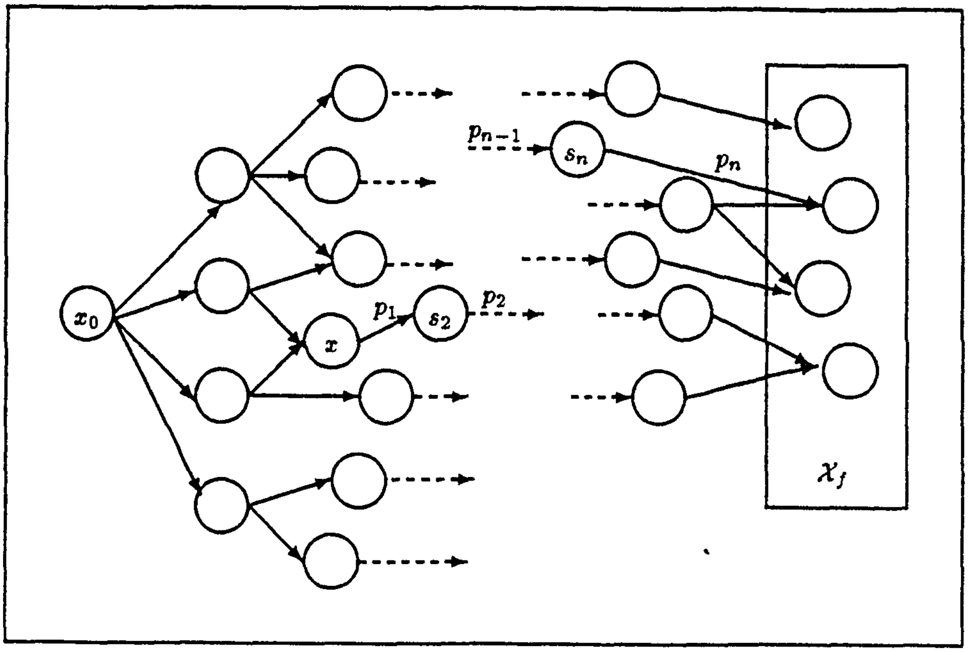
An STN for a query, Q, is a labeled DAG, and can be defined as

$$STN(Q) = (\mathcal{S}, \mathcal{P}, P, \Gamma, x_0, \mathcal{X}_f)$$

where $\mathcal{S}$ is a set of states (nodes); each node contains what remains to be calculated of query Q along with the data structures that can be used to compute the query[2] (i.e., intermediate results, the ELAP, the cache, backlogs). $\mathcal{P}$ is a set of operators which describe the query processing and label the edges of the DAG. P is a mapping: $\mathcal{S} \longrightarrow 2^{\mathcal{P}}$, which maps the state space into the power set space of operations, and describes the set of operations applicable at a given state. Γ is the set of transitions, $\Gamma \subseteq \mathcal{S} \times P(\mathcal{S}) \times \mathcal{S}$; thus, an edge is a triplet, (x_1, p, x_2), containing a start state,

2. Note that no computations are actually carried out. We are merely estimating assumed computations.

Figure 5. An outline of an STN.



a label, and an end state. The last two elements of the equation, $x_0 \in \mathcal{S}$, and $\mathcal{X}_f \subseteq \mathcal{S}$ are the initial and the final states, respectively. The initial state contains the uncomputed query, and a final state contains the computed query, and possibly various intermediate results.

A *plan* for a query, Q, and a state, x, tells which sequence of operators to apply to the partially computed query Q at state x in order to arrive at the final state. If $x \neq x_0$ then the plan is *partial*. If we let $p_1 \circ x$ denote the application of operator p_1 at state x then a plan can be expressed as

$$p_1, p_2, p_3, \ldots, p_n \text{ where } p_n \circ \ldots \circ p_3 \circ p_2 \circ p_1 \circ x \in \mathcal{X}_f$$

We associate a cost C with each plan in the obvious way. First, we define $cost : (\mathcal{S}, P(\mathcal{S}), \mathcal{S}) \longrightarrow [0; \infty)$ to be the cost of applying an operator to a state to get a new state (i.e., the cost of an edge in our DAG). Then the cost of a plan is

$$C(x, p_1, p_2, p_3, \ldots, p_n) = cost(x, p_1, s_2) + cost(s_2, p_2, s_3) + \\ cost(s_3, p_3, s_4) + \ldots + cost(s_n, p_n, x_f)$$

where $x_f \in \mathcal{X}_f$; Figure 5 shows this plan as a part of a larger network.

The minimal cost of a query, Q, is defined as the minimum over all possible plans for Q and x:

$$C_Q(x) = \min\{C(x, p_1, p_2, p_3, \ldots, p_n) \mid p_n \circ \ldots \circ p_3 \circ p_2 \circ p_1 \circ x \in \mathcal{X}_f\}$$

A plan $p_1, p_2, p_3, \ldots, p_n$ for which $C(x, p_1, p_2, p_3, \ldots, p_n) = C_Q(x)$ is *optimal.*

4.2 Plan Selection

Assuming we have costs for all single state transitions, the cheapest query plan in the network can be found by applying dynamic programming techniques. The function $C_Q(x)$ of the previous subsection can be expressed as:

$$C_Q(x) = \min_{p \in P(x)} \{cost(x, p, x') + C_Q(x')\}$$

Dynamic programming is applicable because the cost of a single transition in an STN depends only on local information and not, for example, on the nature of previous transitions that led to the state of the current transition. This has been termed the *separation assumption* (Lafortune and Wong, 1986).

When using dynamic programming, the task of finding a good query plan is conceptually divided into two phases: generation of the STN of the query to be computed; and estimation and selection of the optimal path in the STN. In practice the whole STN need not be computed before phase two is initiated; parts needed during phase two must, however, be made available when needed and, upon completion, all of the STN will be needed. For this reason, dynamic programming requires a relatively large amount of storage space (RND, Sedgewick, 1988). Among the heuristic techniques the A^* algorithm (Rich, 1983) is an alternative, but until an easily computable and precise heuristic function has been found, dynamic programming seems more promising.

To reduce the potentially large search space and improve performance, we introduce pruning rules (Section 6) that specify the function P. They allow us to eliminate paths that are generally not competitive, and therefore limit the search space with little chance of eliminating advantageous plans.

4.3 State and Transition Spaces

We now present the specific design of the type of STN to be used in IM/T. We describe what constitutes a state and which transitions are possible on the states.

4.3.1 State Spaces of IM/T. IM/T generates a separate STN for each query it optimizes, and each STN has its own state space. A state space is a set of states, each consisting of a set of objects. All the types of objects in a state space are stored on secondary memory, and can be read,[3] used, and as a result new objects can be created.

The query of an STN is ultimately defined in terms of a set of *backlogs*. These are part of all the states for that STN. Together with the backlogs, *cached results* constitute the outsets for query computation, and the content of the cache is part of all states. The cache is not changed during plan enumeration and selection, but can be updated when the selected plan is processed. Similarly, the ELAP is part of each state of any STN. The final component of states is *intermediate results*. An intermediate result is any query that can be expressed in terms of backlogs, cached results, and existing intermediate results. Thus, differential files are also intermediate results. Generally a state will contain a set of intermediate results to be used in further computations in order to achieve the evaluation of the query of the STN at hand. With the exception of differential files, such results can later be stored in the cache if they are part of the plan chosen for actual execution. In this case, the ELAP is updated to reflect the new state of the cache. Even if the state of the cache is not changed, the ELAP can be updated with statistics of computed, or estimated, temporary results.

Two states with mutually equivalent objects are identical states.

4.3.2 Transition Space of IM/T. We define the transition space below. In Section 5 we will discuss implementation of the operators of the transition space. The conventional relational operators, projection (π), selection (σ), and equi-join ($\bowtie$) are included. Differential operators are included. In differential computation, previously computed query results are reused in conjunction with differential files to compute desired queries. To more precisely define the differential operators, we first consider differential files in more detail.

Let Q and Q' be query expressions that evaluate to relations with identical schemas. A differential file from Q to Q', $\delta(Q \rightarrow Q')$, satisfies two requirements. First, it consists of an ordered pair of relations with schemas identical to those of Q and Q'.

$$\delta(Q \rightarrow Q') = (\delta^{-}(Q \rightarrow Q'), \delta^{+}(Q \rightarrow Q'))$$

Second, the result of query Q' may be computed from the result of query Q by applying the differential file as follows.

$$Q' = (Q - \delta^{-}(Q \rightarrow Q')) \cup \delta^{+}(Q \rightarrow Q')$$

3. Objects still exist after they have been read.

For this expression, we will use the notation $Q' = \text{DIF}(Q, \delta(Q \rightarrow Q'))$. Now, differential selection, projection, and join may be defined as follows.

$$\begin{aligned}
\text{DIF-}\sigma(\sigma_F(Q), \delta(Q \rightarrow Q')) &= (\sigma_F(Q))' \\
\text{DIF-}\pi(\pi_A(Q), \delta(Q \rightarrow Q')) &= (\pi_A(Q))' \\
\text{DIF-}\bowtie(Q_1 \bowtie Q_2, Q_1, \delta(Q_1 \rightarrow Q_1'), Q_2, \delta(Q_2 \rightarrow Q_2')) &= (Q_1 \bowtie Q_2)'
\end{aligned}$$

Here, we define $(\sigma_F(Q))' = \sigma_F(Q)'$, $(\pi_A(Q))' = \pi_A(Q)'$, and $(Q_1 \bowtie Q_2)' = Q_1' \bowtie Q_2'$. In differential selection, the desired query, $(\sigma_F(Q))'$, may be computed from an already computed query, $\sigma_F(Q)$, and the differential file $\delta(Q \rightarrow Q')$. This contrasts recomputation where first Q' is (re-)computed and then the selection, σ_F, is re-applied.

Operators that provide the differential files used in the differential formulas are included in the transition space.

$$\begin{aligned}
\text{DELTA}(t_1 \rightarrow t_2, B_R) &= \delta(R(t_1) \rightarrow R(t_2)) \\
\text{DELTA-}\sigma(F, \delta(Q \rightarrow Q')) &= \delta(\sigma_F(Q) \rightarrow (\sigma_F(Q))') \\
\text{DELTA-}\pi(A, \delta(Q \rightarrow Q')) &= \delta(\pi_A(Q) \rightarrow (\pi_A(Q))') \\
\text{DELTA-}\bowtie(Q_1, \delta(Q_1 \rightarrow Q_1'), Q_2, \delta(Q_2 \rightarrow Q_2')) &= \delta((Q_1 \bowtie Q_2) \rightarrow (Q_1 \bowtie Q_2)')
\end{aligned}$$

The first of these operators takes a backlog relation, B_R, and two time values, t_1, t_2, as arguments. The result is the differential file that, when applied to $R(t_1)$ results in $R(t_2)$. This is the base case operator. The remaining operators are the step case operators. For example, the DELTA-σ-operator will, given a selection criterion, F, and a differential file from Q to Q', return the differential file from $\sigma_F(Q)$ to $(\sigma_F(Q))'$.

Finally, combined operators are included so that combined operators may be more efficiently processed than sequences of uncombined operators. To illustrate this, assume that we have available the result of the query $\pi_A(\sigma_F(R(t_1)))$ and that we want to compute the query $\pi_A(\sigma_F(R(t_2)))$. With the operators above, we may proceed as follows.

1. Evaluate DELTA$(t_1 \rightarrow t_2, B_R)$ to get $\delta(R(t_1) \rightarrow R(t_2))$.
2. Evaluate DELTA-$\sigma(F, \delta(R(t_1) \rightarrow R(t_2)))$ to get $\delta(\sigma_F(R(t_1)) \rightarrow \sigma_F(R(t_2)))$.
3. Evaluate DIF-$\pi(\pi_A(\sigma_F(R(t_1))), \delta(\sigma_F(R(t_1)) \rightarrow \sigma_F(R(t_2)))$ to obtain the final result.

This implies that the results of both Step 1 and Step 2 are written to disk. A combined operator such as DIF-$\pi\sigma(\pi_A(\sigma_F(Q)), \delta(Q \rightarrow Q'))$ eliminates the storage of the result of Step 2 by processing π_A and σ_F in a single pass. In general, we allow for combining selection and projection with another operator (σ, π, $\bowtie$, or combined) into a combined operator.

4.3.3 Identity Transformations. A user query can be processed in many ways to produce the desired result. During query optimization, equivalence transformation rules for algebra expressions are utilized to enumerate the possible execution orders for a query. We add the following three rules to the ones presented in the literature (Ullman, 1982; Jarke and Koch, 1984; Smith and Chang, 1975).

1. Substituting selection and differential selection.

$$\text{DIF-}\sigma(\sigma_F(Q), \delta(Q \rightarrow Q')) \equiv \sigma_F(\text{DIF}(Q, \delta(Q \rightarrow Q')))$$

2. Substituting projection and differential projection.

$$\text{DIF-}\pi(\pi_A(Q), \delta(Q \rightarrow Q')) \equiv \pi_A(\text{DIF}(Q, \delta(Q \rightarrow Q')))$$

3. Substituting join and differential join.

$$\text{DIF-}\bowtie(Q_1 \bowtie Q_2, Q_1, \delta(Q_1 \rightarrow Q_1'), Q_2, \delta(Q_2 \rightarrow Q_2')) \equiv \\ \text{DIF}(Q_1, \delta(Q_1 \rightarrow Q_1')) \bowtie \text{DIF}(Q_2, \delta(Q_2 \rightarrow Q_2'))$$

The proofs of these equivalences are similar and straightforward. For example, the lefthand side of the third rule is equivalent to $(Q_1 \bowtie Q_2)'$, by definition. This in turn is equivalent to $Q_1' \bowtie Q_2'$. By definition of DELTA, the righthand side of the third rule is equivalent to $Q_1' \bowtie Q_2'$. Thus, the rule follows.

4.4 Using the ELAP for Cache Access

We have included a cache for views in IM/T, and we have defined an ELAP as a "structuring index" on the cache. The role of the ELAP is to allow for efficient identification of cached results that can be used to compute a query at hand.

Let DB be a database instance (i.e., an instance of the backlog store) and Q^c the defining expression of a cached result, then $Q^c(DB)$ is the cached result of Q^c on DB.

The result $Q^c(DB)$ is only useful for the computation of a (sub-)query, Q_s, if the data of $Q_s(DB)$ are all contained in $Q^c(DB)$, and can be extracted from $Q^c(DB)$ using an expression, E, of the query language (Yang and Larson, 1985). If this is the case for any database instance, we say that Q^c *covers* Q_s, $Q_s \sqsubseteq Q^c$.

Coverage is an intensional property. Formally,

$$Q_s \sqsubseteq Q^c \overset{\text{def}}{\Leftrightarrow} \exists E \; (\tilde{Q}_s \equiv E(\tilde{Q}^c))$$

where $\tilde{Q}$ denotes Q where temporal information (time slice) is ignored. Thus, $\sigma_{x \geq 15}(R(t_1)) \sqsubseteq \sigma_{x \geq 10}(R(t_2))$, even if $t_1 \neq t_2$, because $\sigma_{x \geq 15}(R) \equiv \sigma_{x \geq 15}$ $(\sigma_{x \geq 10}(R))$. The covering queries we are most interested in are the ones that are most cheaply modified to the requested query, i.e., the minimal covering queries. Certainly, if $Q_1 \sqsubseteq Q_2 \sqsubseteq Q_3$ then, considering only coverage, we would prefer to use Q_2 instead of Q_3 to compute Q_1.

Orthogonal to the issue of coverage there is the issue of *temporal closeness* which we have disregarded so far. There is both an intensional and an extensional aspect. We address the intensional aspect first. When we have retrieved a result from the cache it might not reflect the state we are interested in. If we let $Q_s = \sigma_{x \geq 10}(R(t_1))$ and let $Q_1^c = \sigma_{x \geq 10}(R(t_a))$ then the two queries are identical under coverage, but if $t_1 \neq t_a$, the operator DIF (probably) still needs to be applied to Q_1^c and an appropriate differential file to make it correctly reflect the desired state. Assume the existence of $Q_2^c = \sigma_{x \geq 10}(R(t_b))$. If the temporal expressions t_a and t_b are both fixed then we would choose Q_1^c if t_a is closer to t_1 than is t_b otherwise, we would choose Q_2^c. The concept of closeness is defined in terms of the cost of the differential computation that has to be carried out in order to reach the desired state, and it depends on the size of the portions of the associated backlog that has to be processed. The distance between time stamps is an intensional property which can be used for comparing closeness. However, if $t_a \leq t_1 \leq t_b$ or $t_b \leq t_1 \leq t_a$, the distance between time stamps is not a reliable means of comparison.

The extensional aspect of closeness is important because cache entries generally get outdated (because of the variable *NOW*). In the context of time dependent views, it is not sufficient only to look at the intensions of queries as we did above where we compared t_1, t_a, and t_b. For example, if $Q_s = \sigma_{x \geq 10}(R(t_1))$, and the cache contains $Q_1^c = \sigma_{x \geq 10}(R(t_1))$ and $Q_2^c = \sigma_{x \geq 10}(R(t_2))$, where $t_1 \neq t_2$ then Q_2^c still can be more useful than Q_1^c. This is so because t_1 could be time dependent and Q_1^c could be very outdated. Outdatedness of a cached query result is defined as the closeness between the defining query expression at the time it was computed and the current defining query expression.

For each cached result the ELAP stores the value of the variable *NOW* at the time when the result was computed so that the states of cached results can be inferred without actually accessing them. Also the ELAP holds statistics that can help estimate the outdatedness of results (i.e., estimate the number of change requests between two points in time and the cost of processing them appropriately).

Figure 6. Implementation of operators of STNs.

$$\left\{\begin{array}{c} \left\{\begin{array}{c}\sigma\\ \pi\\ \bowtie\end{array}\right\} \times \left\{\begin{array}{c}\text{data}\\ \text{pointer}\end{array}\right\} \\ \text{DELTA} \times \left\{\begin{array}{c}\text{INCR}\\ \text{DECR}\end{array}\right\} \times \text{data} \\ \left\{\begin{array}{c}\text{DELTA-}\sigma\\ \text{DELTA-}\pi\\ \text{DELTA-}\bowtie\\ \text{DIF}\\ \text{DIF-}\sigma\\ \text{DIF-}\pi\\ \text{DIF-}\bowtie\end{array}\right\} \times \left\{\begin{array}{c}\text{data}\\ \text{pointer}\end{array}\right\} \end{array}\right\}$$

5. Implementation of Operators of STNs

Here we discuss the operators in more detail. Initially, we outline the different cases to consider. Based on these, we discuss alternatives for implementation of the operators.

5.1 Overview of Operators

The operators considered are outlined in Figure 6. The figure has 22 entries, each corresponding to a separate case. In IM/T results can be stored as either actual data or pointers that point to the data. The entries "data" and "pointer" indicate the type of arguments. All operators must work on both kinds of arguments, with one exception: The DELTA operator in both the incremental and the decremental case is applied to a backlog which is a data argument. In the Figure 6, the type of the result returned by an operator is assumed to be the same as the type of the arguments. However, if the arguments are data, both data and pointer results are possible, the only restriction being that differential files are assumed to be data. This gives an additional seven cases (i.e., three for σ, π, and $\bowtie$ with data arguments;

four for DIF, DIF-σ, DIF-π, and DIF-$\bowtie$ with data arguments).

As the first six entries, we find the ordinary operators σ, π, and $\bowtie$.[4] These operators have their standard semantics and can be implemented as suggested in the literature (e.g., Selinger et al., 1979; Shapiro, 1986.)

The remaining sixteen cases concern the new operators. The operator DELTA incrementally or decrementally processes sequences of change requests stored in backlogs to get differential files. The three remaining cases of the DELTA operator are the computations of differential files of relations from which they are derived by either projection, selection, or join.

The four DIF operators differentially update a stored result to correctly reflect a desired state. The operators differ on how the outset is related to the differential file(s) to be used. It is possible to use the differential file of a relation from which the outset is derived by a projection (including the identity projection) or a selection. The differential files of relations from which the outset is derived by a join can be used also.

Finally, selections and projections can be done on the fly, meaning that a selection and a projection can be performed interleaved with another operator (selection, projection, or join) in a single pass without storage of intermediate results.

In the following, we will generally consider only the cases where the operators take data arguments and produce data results.

5.2 Selection, Projection, and Join

The traditional relational algebra operators, selection, projection, and join, can be applied to any relation, including differential files, $(\delta^-(Q \rightarrow Q'), \delta^+(Q \rightarrow Q'))$. The expression F of the selection operator, $\sigma_F(Q)$, can contain a conjunction of selection criteria of the form `Att_Name op Att_Name` or `Att_Name op Value` where op is one of $=, <, >, \geq, \leq, \neq, \not<, \not>, \not\geq$, or $\not\leq$, and `Att_Name` is an attribute identifier of the relation valued expression Q. The most advantageous implementation of selection depends on numerous factors and has been addressed already in many settings. Consequently we will not address it here.

The projection expression, A, of $\pi_A(Q)$ is any subset of attributes of Q. When we do differential computations, we would like to be able to distribute projections over difference. In order to make this legal, we must at all times make sure that unique identification of tuples is possible. We choose to do this by always retaining the primary key of relations, remembering whether it was removed by projection

4. In the following, $\bowtie$ denotes equi-join.

or not.

The equi-join operator, $Q_1 \bowtie_F Q_2$, can be used on any two query expressions. The condition F is a list of elements of the form `Att_Name_1 = Att_Name_2` where `Att_Name_1` is an attribute of relation Q_1 (Q_2) and `Att_Name_2` is an attribute of the expression Q_2 (Q_1). Several ways have been suggested for doing binary joins, e.g., Hash-Join, Nested-Loop-Join, Sort-Merge-Join. For a thorough treatment, see Shapiro (1986).

Finally, selection and projection can be combined with any operator (possibly combined) to form a combined operator.

5.3 Computing Differential Files

Here we discuss each of the DELTA operators. Recall that a differential file from Q to Q', both query expressions with identical schemas, is denoted $\delta(Q \rightarrow Q')$ so that $\delta(Q \rightarrow Q') = (\delta^-(Q \rightarrow Q'), \delta^+(Q \rightarrow Q'))$, and $Q' = (Q - \delta^-(Q \rightarrow Q')) \cup \delta^+(Q \rightarrow Q')$.

5.3.1 The Base Cases. The operator DELTA$(t_a \rightarrow t_x, B_R)$ generates a differential file, $\delta(R(t_a) \rightarrow R(t_x))$, directly from a backlog, B_R. This operator differs from the three other DELTA operators in that a list of change requests in a backlog is the argument.

If $t_a < t_x$ then the requested state of R is a future state relative to its current state, and we are in the incremental case. If $t_a > t_x$ then we are in the decremental case.

The construction procedure for $\delta^+(R(t_a) \rightarrow R(t_x))$ and $\delta^-(R(t_a) \rightarrow R(t_x))$ starts with the initialization of these to empty relations. The schema of $\delta^+(R(t_a) \rightarrow R(t_x))$ is the same as that of R, and the schema of $\delta^-(R(t_a) \rightarrow R(t_x))$ only contains the primary key attribute of that of R.[5] Then we process change requests from the outset in the direction of t_x until the next change request to be processed has a time stamp that is not in the half-open interval from t_a to, and including, t_x.

Each request is projected to remove superfluous attribute values. Assuming that we are in the incremental case, insertion requests go into $\delta^+(R(t_a) \rightarrow R(t_x))$ which optionally can be kept sorted on key values, or/and an (hash) index on key values can be maintained. A deletion request refers to either a tuple in the outset or to a tuple in

5. Only the key is needed in an actual implementation, but in algebra expressions we assume for simplicity that the schema is the same as that of R.

$\delta^+(R(t_a) \rightarrow R(t_x))$.[6] First $\delta^+(R(t_a) \rightarrow R(t_x))$ is searched for a tuple matching the deletion request, and if a match is found, then the request is disregarded, and the matching tuple of the current $\delta^+(R(t_a) \rightarrow R(t_x))$ is deleted, because the net effect is that no change takes place; otherwise, the deletion request goes into $\delta^-(R(t_a) \rightarrow R(t_x))$. Note that no action was taken when we encountered an insertion request of a previously encountered deletion request. Such corresponding deletion and insertion requests must be carried out because they update implicit time stamp attributes of base relations; such attributes are hidden, but can be seen by explicit projections. Here, we ignore these implicit attributes. Tuples of $\delta^+(R(t_a) \rightarrow R(t_x))$ and $\delta^-(R(t_a) \rightarrow R(t_x))$ are written to secondary memory one page at a time. Note that there are no references from $\delta^-(R(t_a) \rightarrow R(t_x))$ to $\delta^+(R(t_a) \rightarrow R(t_x))$, making the sequence of operation in the formula above valid in the sense that the outcome of DIF-$(R(t_a), \delta(R(t_a) \rightarrow R(t_x)))$ is, in fact, $R(t_x)$. Also note that there can be references from $\delta^+(R(t_a) \rightarrow R(t_x))$ to $\delta^-(R(t_a) \rightarrow R(t_x))$, making the sequence of operation in the formula the only valid one.

When there are no more change requests, both differentials are stored sorted on key values, and the optional index on $\delta^+(R(t_a) \rightarrow R(t_x))$ is deleted.

In the decremental case, the only change is that deletion requests assume the role of insertion requests, and vice versa.

5.3.2 The Step Cases. Now, we consider the cases where a differential file of a result is constructed from the differential file of another result. In DELTA-$\sigma(F, \delta(Q \rightarrow Q'))$, the operator constructs the differential file from $\sigma_F(Q)$ to $(\sigma_F(Q))'$ using the differential file from Q to Q', $\delta(Q \rightarrow Q')$, where Q denotes any query expression. This is just a selection:

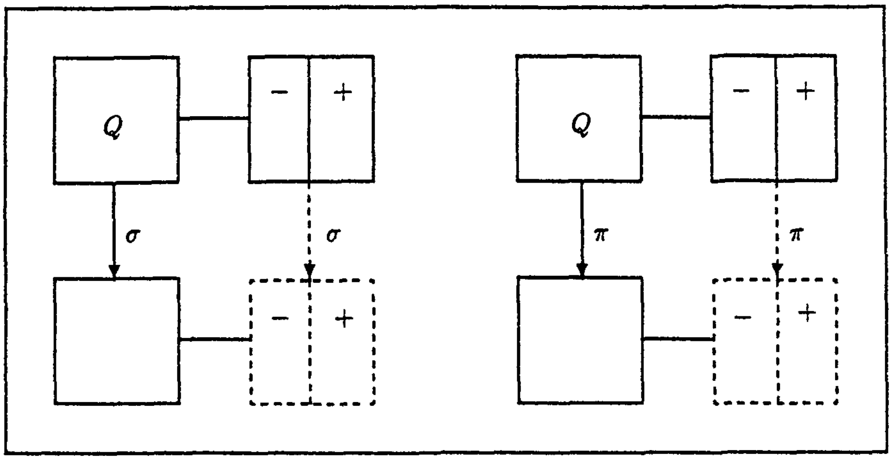
$$\text{DELTA-}\sigma(F, \delta(Q \rightarrow Q')) = \sigma_F(\delta(Q \rightarrow Q')) = \\ (\sigma_F(\delta^-(Q \rightarrow Q')), \sigma_F(\delta^+(Q \rightarrow Q')))$$

Claiming that this correctly computes $\delta(\sigma_F(Q) \rightarrow (\sigma_F(Q))')$ is equivalent to claiming that the following expression correctly computes $(\sigma_F(Q))'$.

$$(\sigma_F(Q) - \sigma_F(\delta^-(Q \rightarrow Q'))) \cup \sigma_F(\delta^+(Q \rightarrow Q'))$$

6. Note that eagerly maintained current states of user-defined rollback relations allow for checking that deletions and insertions actually make sense, i.e., that deletions actually delete something existing and, conversely, that insertions actually insert something not already existing. These are system enforced integrity constraints.

Figure 7. The DELTA-σ and DELTA-π operators.



This expression is equivalent to the following.

$$\sigma_F((Q - \delta^-(Q \rightarrow Q')) \cup \delta^+(Q \rightarrow Q'))$$

Correctness follows as this is equivalent to $\sigma_F(Q')$ which, in turn, is equivalent to $(\sigma_F(Q))'$.

Next, in DELTA-$\pi(A, \delta(Q \rightarrow Q'))$, we make a projection:

$$\text{DELTA-}\pi(A, \delta(Q \rightarrow Q')) = \pi_A(\delta(Q \rightarrow Q')) = (\pi_A(\delta^-(Q \rightarrow Q')),\\ \pi_A(\delta^+(Q \rightarrow Q'))$$

Remember that key information is retained to overcome the problem of indistinguishable tuples when distributing a projection over a difference. The proof of correctness is similar to that of the DELTA-σ operator. Figure 7 is a schematical representation of the DELTA-σ and DELTA-π operators.

The last case is the join: DELTA-$\bowtie$ $(Q_1, \delta(Q_1 \rightarrow Q_1'), Q_2, \delta(Q_2 \rightarrow Q_2'))$. To construct the differential file of $Q_1 \bowtie Q_2$, we need both Q_1, Q_2, $\delta(Q_1 \rightarrow Q_1')$, and $\delta(Q_2 \rightarrow Q_2')$. In order to explain the derivation of the formula for computing $\delta((Q_1 \bowtie Q_2) \rightarrow (Q_1 \bowtie Q_2)')$, consider the following three equalities:

$$\begin{aligned}
Q_1' \bowtie Q_2' &= (Q_1 \bowtie Q_2)' && (1)\\
(Q_1 \bowtie Q_2)' &= [(Q_1 \bowtie Q_2) - \delta^-((Q_1 \bowtie Q_2) \rightarrow (Q_1 \bowtie Q_2)')] && (2)\\
&\quad \cup \delta^+((Q_1 \bowtie Q_2) \rightarrow (Q_1 \bowtie Q_2)')\\
Q_1' \bowtie Q_2' &= [((Q_1 - \delta^-(Q_1 \rightarrow Q_1')) \cup \delta^+(Q_1 \rightarrow Q_1')] \bowtie && (3)\\
&\quad [((Q_2 - \delta^-(Q_2 \rightarrow Q_2')) \cup \delta^+(Q_2 \rightarrow Q_2')]
\end{aligned}$$

We derive $\delta((Q_1 \bowtie Q_2) \rightarrow (Q_1 \bowtie Q_2)')$ by transforming the right hand side of equality (3) into an equivalent expression of the form $[(Q_1 \bowtie Q_2) - \boxed{x}\,] \cup \boxed{y}$. Then, by equalities (1) and (2), $(\delta^-((Q_1 \bowtie Q_2) \rightarrow (Q_1 \bowtie Q_2)'), \delta^+((Q_1 \bowtie Q_2) \rightarrow (Q_1 \bowtie Q_2)')) = (\boxed{x}, \boxed{y})$.

To do the transformation, we need two transformation rules:

$$\begin{aligned}(Q_1 \cup Q_2) \bowtie Q_3 &\equiv (Q_1 \bowtie Q_3) \cup (Q_2 \bowtie Q_3)\\ (Q_1 - Q_2) \bowtie Q_3 &\equiv (Q_1 \bowtie Q_3) - (Q_2 \bowtie Q_3)\end{aligned}$$

To derive the first, observe that $(Q_1 \cup Q_2) \times Q_3 \equiv (Q_1 \times Q_3) \cup (Q_2 \times Q_3)$. Because, in addition, $Q_1 \bowtie Q_2 \equiv \sigma_F(Q_1 \times Q_2)$ where F is the equi-join condition, then

$$\begin{aligned}(Q_1 \cup Q_2) \bowtie Q_3 &\equiv \sigma_F[(Q_1 \cup Q_2) \times Q_3] \equiv \sigma_F[(Q_1 \times Q_3) \cup (Q_2 \times Q_3)]\\ &\equiv \sigma_F(Q_1 \times Q_3) \cup \sigma_F(Q_2 \times Q_3) \equiv (Q_1 \bowtie Q_3) \cup (Q_2 \bowtie Q_3)\end{aligned}$$

The second is proven as follows. First, assume that $x \in (Q_1 - Q_2) \bowtie Q_3$; then, we prove that $x \in (Q_1 \bowtie Q_3) - (Q_2 \bowtie Q_3)$. The element x is of the form (x_1, x_2) where $x_1 \in (Q_1 - Q_2)$ and $x_2 \in Q_3$. Further, $x_1 \in Q_1$ and $x_1 \notin Q_2$. Hence, $(x_1, x_2) \in Q_1 \bowtie Q_3$, and $(x_1, x_2) \notin Q_2 \bowtie Q_3$.

Second, we assume the converse and prove that $x \in (Q_1 - Q_2) \bowtie Q_3$. Here, $x \in Q_1 \bowtie Q_3$, and $x \notin Q_2 \bowtie Q_3$. Consequently, $x_1 \in Q_1$, and $x_2 \in Q_3$, and also $x_1 \notin Q_2$. But then $x_1 \in Q_1 - Q_2$.

Using the abbreviations $\delta_1^{+/-}$ and $\delta_2^{+/-}$ for $\delta^{+/-}(Q_1 \rightarrow Q_1')$ and $\delta^{+/-}(Q_2 \rightarrow Q_2')$, respectively, we now have

$$\begin{aligned}
&[(Q_1 - \delta_1^-) \cup \delta_1^+] \bowtie [(Q_2 - \delta_2^-) \cup \delta_2^+]\\
&\equiv \{(Q_1 - \delta_1^-) \bowtie [(Q_2 - \delta_2^-) \cup \delta_2^+]\} \cup \{\delta_1^+ \bowtie [(Q_2 - \delta_2^-) \cup \delta_2^+]\}\\
&\equiv \{[(Q_1 - \delta_1^-) \bowtie (Q_2 - \delta_2^-)] \cup [(Q_1 - \delta_1^-) \bowtie \delta_2^+]\} \cup\\
&\quad \{[\delta_1^+ \bowtie (Q_2 - \delta_2^-)] \cup (\delta_1^+ \bowtie \delta_2^+)\}\\
&\equiv \{\{[Q_1 \bowtie (Q_2 - \delta_2^-)] - [\delta_1^- \bowtie (Q_2 - \delta_2^-)]\} \cup [(Q_1 \bowtie \delta_2^+) -\\
&\quad (\delta_1^- \bowtie \delta_2^+)]\} \cup \{[(\delta_1^+ \bowtie Q_2) - (\delta_1^+ \bowtie \delta_2^-)] \cup (\delta_1^+ \bowtie \delta_2^+)\}\\
&\equiv \{\{[(Q_1 \bowtie Q_2) - (Q_1 \bowtie \delta_2^-)] - [(\delta_1^- \bowtie Q_2) - (\delta_1^- \bowtie \delta_2^-)]\} \cup\\
&\quad [(Q_1 \bowtie \delta_2^+) - (\delta_1^- \bowtie \delta_2^+)]\} \cup \{[(\delta_1^+ \bowtie Q_2) - (\delta_1^+ \bowtie \delta_2^-)] \cup (\delta_1^+ \bowtie \delta_2^+)\}\\
&\equiv \{(Q_1 \bowtie Q_2) - (Q_1 \bowtie \delta_2^-) - [(\delta_1^- \bowtie Q_2) - (\delta_1^- \bowtie \delta_2^-)]\} \cup\\
&\quad [(Q_1 \bowtie \delta_2^+) - (\delta_1^- \bowtie \delta_2^+)] \cup [(\delta_1^+ \bowtie Q_2) - (\delta_1^+ \bowtie \delta_2^-)] \cup (\delta_1^+ \bowtie \delta_2^+)
\end{aligned}$$

The two last right hand sides contain different expressions for the differential file of a join. For example, using the last one, we have

$$\begin{aligned}
&\text{DELTA-}\bowtie(Q_1, \delta(Q_1 \rightarrow Q_1'), Q_2, \delta(Q_2 \rightarrow Q_2')) \\
&\equiv (\delta^-((Q_1 \bowtie Q_2) \rightarrow (Q_1 \bowtie Q_2)'), \delta^+((Q_1 \bowtie Q_2) \rightarrow (Q_1 \bowtie Q_2)')) \\
&\equiv ([\underbrace{Q_1 \bowtie \delta_2^-}_{a} \cup \underbrace{(\delta_1^- \bowtie Q_2) - (\delta_1^- \bowtie \delta_2^-)}_{b}], \\
&\quad [\underbrace{(Q_1 \bowtie \delta_2^+) - (\delta_1^- \bowtie \delta_2^+)}_{x} \cup \underbrace{(\delta_1^+ \bowtie Q_2) - (\delta_1^+ \bowtie \delta_2^-)}_{y} \cup \underbrace{\delta_1^+ \bowtie \delta_2^+}_{z}])
\end{aligned}$$

The components of $\delta^-((Q_1 \bowtie Q_2) \rightarrow (Q_1 \bowtie Q_2)')$ are: (a) the deletions to $Q_1 \bowtie Q_2$ due to deletions from Q_2; and (b) the deletions to $Q_1 \bowtie Q_2$ due to deletions from Q_1, but with overlapping deletions (i.e., $\delta^-(Q_1 \rightarrow Q_1') \bowtie \delta^-(Q_2 \rightarrow Q_2')$) removed.

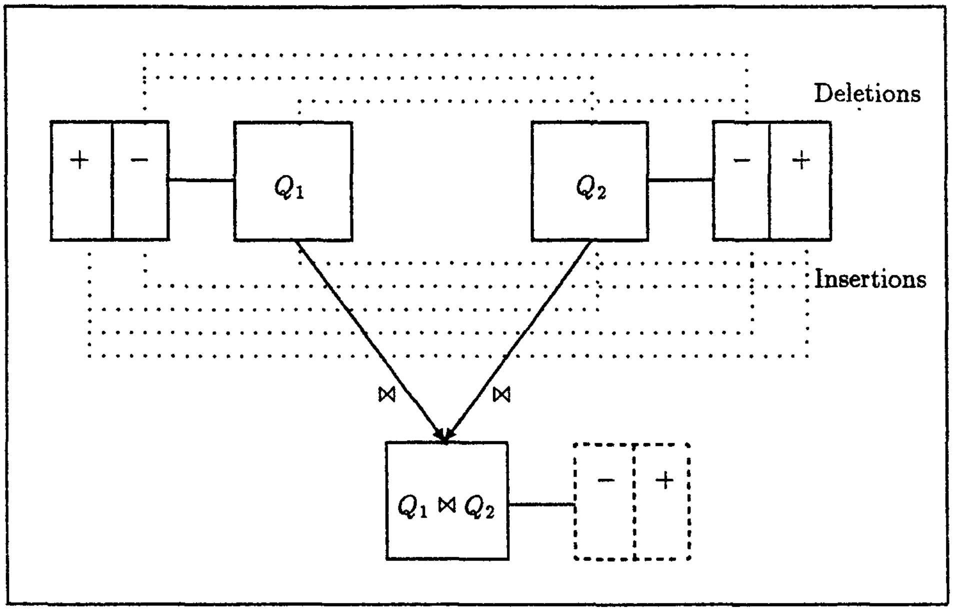
The components of $\delta^+((Q_1 \bowtie Q_2) \rightarrow (Q_1 \bowtie Q_2)')$ are: (x) insertions to the outset due to tuples from Q_1 matching insertions to Q_2, but not including tuples due to matches between insertions to Q_2 and deletions to Q_1; (y) a component symmetric, in Q_1 and Q_2, to (x); (z) insertions to the outset due to matches between insertions in Q_1 and insertions in Q_2. Figure 8 shows all the constituent joins of $\delta^-((Q_1 \bowtie Q_2) \rightarrow (Q_1 \bowtie Q_2)')$ and $\delta^+((Q_1 \bowtie Q_2) \rightarrow (Q_1 \bowtie Q_2)')$ by means of dotted lines connecting two relations.

There are no deletions of insertions in the differential file of a join, DELTA-$\bowtie (Q_1, \delta(Q_1 \rightarrow Q_1'), Q_2, \delta(Q_2 \rightarrow Q_2'))$. To see why, observe that neither of the two components (a and b) for $\delta^-((Q_1 \bowtie Q_2) \rightarrow (Q_1 \bowtie Q_2)')$ overlap any of the three components of $\delta^-((Q_1 \bowtie Q_2) \rightarrow (Q_1 \bowtie Q_2)')$ (x, y, and z). (a, x) and (a, z): disjoint because $\delta^-(Q_2 \rightarrow Q_2')$ and $\delta^+(Q_2 \rightarrow Q_2')$ are disjoint. (b, y) and (b, z): disjoint because $\delta^-(Q_1 \rightarrow Q_1')$ and $\delta^+(Q_1 \rightarrow Q_1')$ are disjoint. (a, y): disjoint because $\delta^-(Q_2 \rightarrow Q_2')$ and $Q_2 - \delta^-(Q_2 \rightarrow Q_2')$ are disjoint. (b, x): disjoint because $\delta^-(Q_1 \rightarrow Q_1')$ and $Q_1 - \delta^-(Q_1 \rightarrow Q_1')$ are disjoint.

As shown, the differential of a join is a complex query and it can be computed in many ways (Blakely et al., 1986). Techniques from multiple query optimization can be exploited (Jarke, 1984; Kim, 1984; Chakravarthy and Minker, 1986; Sellis, 1986, 1988*b*). For example, keeping all six argument relations sorted, joins can be done interleaved and pagewise (pipe-line join).

It is straightforward to implement operators such as DELTA-$\pi\sigma(A, F, \delta(Q \rightarrow Q'))$ where projection and selection is combined. This is done using combined operators of the previous subsection. "Combined" generation of differential files directly from change requests and selections/projections is possible also.

Figure 8. Computation of differentials of joins.



5.4 Incrementing/Decrementing Relations

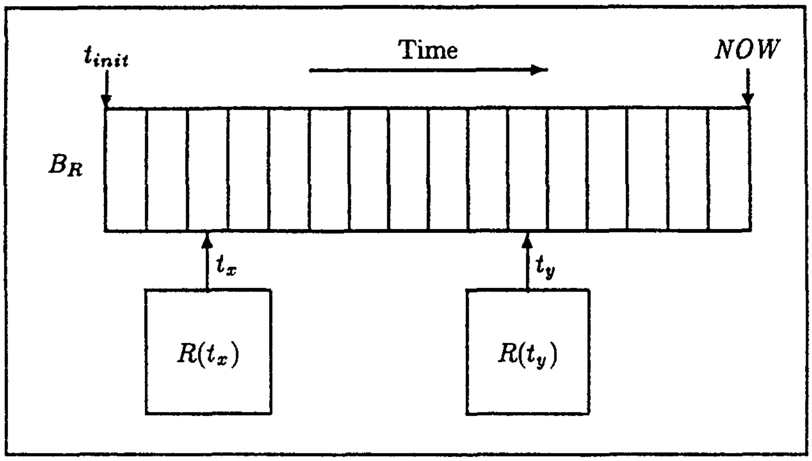
Now we discuss the implementation of the four operators for differential computation in turn.

5.4.1 Time Slicing Base Relations—The Base Cases. For the DIF operator we will investigate two cases. First we consider the special case of computing DIF$(R(t_x), \delta(R(t_x) \rightarrow R(t_y)))$ where we use the backlog B_R directly as an alternative to first computing $\delta(R(t_x) \rightarrow R(t_y))$. Second, we consider the general case, DIF$(Q, \delta(Q \rightarrow Q'))$.

The first case is illustrated in Figure 9. Note that both incremental and decremental computation are always possible (with $t_x = t_{init}$ and t_x =*NOW*, respectively).

In this case, change requests are processed one at a time towards the requested state from the outset until the time stamp of the next change request to be considered exceeds the time of the desired state. The result of an insertion request is that the tuple of the request is entered into the current outset, and the result of a deletion request is that the tuple identified by the request is removed from the current state.

Figure 9. Time-slicing a base relation.

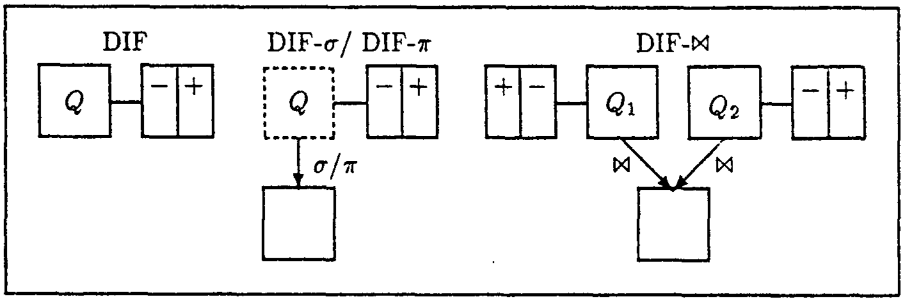


In the general case, DIF$(Q, (\delta^-(Q \rightarrow Q'), \delta^+(Q \rightarrow Q')))$, we initially sort $\delta^-(Q \rightarrow Q')$ and $\delta^+(Q \rightarrow Q')$ if they were not sorted already. Both δ-files are then simultaneous "merged" with the outset: first a page of deletions is read, then the first relevant page of the outset and the first relevant page of the insertions are read. Deletions are performed on the outset first, then relevant insertions are performed. Whenever a page is totally read, the next page of the relation is read. In the case of the outset, processed pages are written, and only pages that are relevant for the deletions are read (irrelevant pages can be considered processed and written already). When there are neither deletions nor insertions left, the processing terminates. Following this procedure, pages of the three relations are only read once, and irrelevant pages of the outset need not be read at all.

When computing DIF$(R(t_x), \delta(R(t_x) \rightarrow R(t_y)))$ we use the characteristics of the arguments (i.e., the size of the outset used) and the differential file, as criteria for chosing between the first and a variation of the second strategy. The framework includes a component that, given the name of a backlog and a start and an end time, returns estimates: the number of insertions, the total number of change requests, and the number of deletions of insertions. The input to the component is produced during non-eager processing of change requests. If the first strategy is used, counts of insertions and deletions are used; if the second strategy is used, again counts of insertions and deletions are available, but so is also the final number of insertions. How these inputs are most efficiently used to generate the output is a topic of current research.

The first strategy is advantageous if the total number of change requests to be processed is low. The choice of keeping $\delta^+(Q \rightarrow Q')$ sorted or not depends on

Figure 10. Differential selection, projection, and join.



the number of insertions into $\delta^+(Q \rightarrow Q')$ compared to the number of deletions to be processed against $\delta^+(Q \rightarrow Q')$. If sorting is adopted, insertion has an overhead, and if not, then search for deletions must be done by sequential scan.

5.4.2 The Step Cases. The differential selection and projection operators, DIF-σ $(\sigma_F(Q), \delta(Q \rightarrow Q'))$ and DIF-$\pi(\pi_A, \delta(Q \rightarrow Q'))$, respectively, may be computed as follows.

$$\begin{aligned}
&\text{DIF-}\sigma(\sigma_F(Q), \delta(Q \rightarrow Q')) \\
&\quad = \ (\sigma_F(Q) - \sigma_F(\delta^-(Q \rightarrow Q'))) \cup \sigma_F(\delta^+(Q \rightarrow Q')) \\
&\text{DIF-}\pi(\pi_A(Q), \delta(Q \rightarrow Q')) \\
&\quad = \ (\pi_A(Q) - \pi_A(\delta^-(Q \rightarrow Q'))) \cup \pi_A(\delta^+(Q \rightarrow Q'))
\end{aligned}$$

The correctness of these observations follows from the observations $\sigma_F(\delta(Q \rightarrow Q')) = \delta(\sigma_F(Q) \rightarrow \sigma_F(Q)')$ and $\pi_A(\delta(Q \rightarrow Q')) = \delta(\pi_A(Q) \rightarrow \pi_A(Q)')$.

Differential selection, projection, and join are illustrated in Figure 10. For each of the operators, the arguments are shown. The broken box of DIF-σ/DIF-π is not an argument—it is present only to indicate the relationship between the arguments.

The final case is DIF-$\bowtie$ $(Q_1 \bowtie Q_2, Q_1, \delta(Q_1 \rightarrow Q_1'), Q_2, \delta(Q_2 \rightarrow Q_2'))$, the differential join. From Subsection 5.3, we have (again, we abbreviate $\delta^{+/-}(Q_1 \rightarrow Q_1')$ and $\delta^{+/-}(Q_2 \rightarrow Q_2')$ by $\delta_1^{+/-}$ and $\delta_2^{+/-}$, respectively).

$$\begin{aligned}
&DIF(Q_1 \bowtie Q_2, Q_1, \delta_{Q_1}, Q_2, \delta_{Q_2}) \\
&\quad \equiv \ (Q_1 \bowtie Q_2) - \underbrace{(Q_1 \bowtie \delta_2^-)}_{1} - \underbrace{[(\delta_1^- \bowtie Q_2) - (\delta_1^- \bowtie \delta_2^-)]}_{2} \cup \\
&\qquad [(Q_1 \bowtie \delta_2^+) - (\delta_1^- \bowtie \delta_2^+)] \cup [(\delta_1^+ \bowtie Q_2) - (\delta_1^+ \bowtie \delta_2^-)] \cup (\delta_1^+ \bowtie \delta_2^+)
\end{aligned}$$

Let us consider processing of the deletions to the outset. The two components can be explained as follows.

1. $Q_1 \bowtie \delta^-(Q_2 \rightarrow Q_2')$ are all the deletions from the outset due to deletions to Q_2;

2. $(\delta^-(Q_1 \rightarrow Q_1') \bowtie Q_2) - (\delta^-(Q_1 \rightarrow Q_1') \bowtie \delta^-(Q_2 \rightarrow Q_2'))$ are all the deletions to the outset due to deletions to Q_1 with duplicate deletions due to overlaps between $\delta^-(Q_1 \rightarrow Q_1')$ and $\delta^-(Q_2 \rightarrow Q_2')$ and already included in (1) removed.

The overlaps can be ignored without affecting the correctness of the final result, and the deletions represented by the two remaining terms can be performed using only $Q_1 \bowtie Q_2$, $\delta^-(Q_1 \rightarrow Q_1')$, and $\delta^-(Q_2 \rightarrow Q_2')$. A tuple of the outset is of the form (x_{Q_1}, x_{Q_2}) where x_{Q_1} is a tuple compatible with Q_1 and x_{Q_2} is a tuple compatible with Q_2. Tuples of $Q_1 \bowtie Q_2$ where x_{Q_2} match a tuple in $\delta^-(Q_2 \rightarrow Q_2')$ are simply deleted; similarly tuples where x_{Q_1} match a tuple in $\delta^-(Q_1 \rightarrow Q_1')$ are deleted.

Now, let us turn to the insertions. It is instructive to reformulate the expression for $(Q_1 \bowtie Q_2)'$ as follows (with $\delta^-((Q_1 \bowtie Q_2) \rightarrow (Q_1 \bowtie Q_2)')$ abbreviated by δ_{12}^-).

$$\begin{aligned}
&(Q_1 \bowtie Q_2)' \\
&\quad \equiv (Q_1 - \delta_1^-) \bowtie (Q_2 - \delta_2^-) \cup [(Q_1 - \delta_1^-) \bowtie \delta_2^+] \cup \{\delta_1^+ \bowtie [(Q_2 - \delta_2^-) \cup \delta_2^+]\} \\
&\quad \equiv Q_1 \bowtie Q_2 - \delta_{12}^- \cup [(Q_1 - \delta_1^-) \cup \delta_1^+ - \delta_1^+] \bowtie \delta_2^+] \cup \{\delta_1^+ \bowtie [(Q_2 - \delta_2^-) \cup \delta_2^+]\} \\
&\quad \equiv Q_1 \bowtie Q_2 - \delta_{12}^- \cup \\
&\qquad \{[(Q_1 - \delta_1^-) \cup \delta_1^+] \bowtie \delta_2^+\} - [\delta_1^+ \bowtie \delta_2^+] \cup \{\delta_1^+ \bowtie [(Q_2 - \delta_2^-) \cup \delta_2^+]\} \\
&\quad \equiv Q_1 \bowtie Q_2 - \delta_{12}^- \cup \underbrace{\{[(Q_1 - \delta_1^-) \cup \delta_1^+] \bowtie \delta_2^+\}}_{1} \cup \underbrace{[\delta_1^+ \bowtie (Q_2 - \delta_2^-)]}_{2}
\end{aligned}$$

The insertion, $\delta^+((Q_1 \bowtie Q_2) \rightarrow (Q_1 \bowtie Q_2)')$, is now defined by two joins. The first (1) has $\delta^+(Q_2 \rightarrow Q_2')$ as one argument, and the second (2) has $\delta^+(Q_1 \rightarrow Q_1')$ as one argument. This explains the superiority of differential computation when differentials are small and relations large because in such cases an expensive join of two large relations, Q_1' and Q_2', is avoided and two joins of a small relation with a large relation is done instead.[7]

7. Differential computation and recomputation both involve additional processing apart from joins, but, because join is the most expensive operation, we ignore this.

Algorithms, costs, and efficient implementation of incremental join for pointer views in the ADMS system are discussed in detail in Stamenas (1989) and Roussopoulous (1991).

6. Pruning the Search Space

We already have presented a complete framework for query optimization. Here we introduce the concept of pruning a STN. Pruning is a means of further optimization of plan selection. The purpose is to reduce the sizes of the STNs generated without leaving out promising query plans. Reduced STNs mean reduced costs of estimating costs of single transitions and a smaller argument of the dynamic programming algorithm which therefore executes more efficiently. The purpose of introducing the mapping P in the definition of an STN was exactly to be able to include pruning into the framework. The rules of this section restrict the number of possible transitions at a state.

The rules below illustrate the kind of rules that can be integrated into IM/T. Rules from standard query optimization (Ullman, 1982) can be applied, too.

Rule 1. *Only apply a differential to its outset if exactly the selections/projections performed on the outset have been performed on the differential, too.* Obeying this rule will ensure that selections/projections are done on only the outset or the differential, and never on the updated outset. This is reasonable because at least the differential can be assumed to be much smaller than the updated outset.

Rule 2. *Apply operators as early as possible.* If the arguments in state x_b of an operation p (transforming x_b into x_c) are present in an predecessor state, x_a of x_b then p should be applied to x_a instead of to x_b.

Rule 3. *Only compute a differential of an outset, if the outset already exists.* Both sequences are possible, but an STN should only include one of them, and a differential is not useful if the outset is not available.

Rule 4. *Application of maximal combined operators is preferable to the sequential application of the constituent operators of the combined operators.*

Rule 5. *Only use the smallest cached result out of covering results equally outdated with respect to the desired state.* This and the following rule attempt to consider only the most promising cached results during generation af an STN.

Rule 6. *Only use the least outdated cached result out of covering results of equal size.*

7. Conclusion and Future Research

Extending the relational model to automatically record transaction time is not a new idea, but implementing the extended model by storing the complete history of change in relation backlogs is. Such an implementation will support not only queries on previous database states, but queries on the nature of change itself.

We expect queries on the nature of change to play a key role in future information systems. With ever-increasing amounts of constantly changing information, it will be impossible for an individual user to digest all the information that pertains to a given situation and stay abreast of all its changes. We will see applications where the user is not interested in the current state of the database and the changes made to it, as long as they are both normal. On the other hand, if the current state of the database or the change made to it is abnormal, then the user is interested and must be notified. The price paid for the added functionality is a substantial increase of space consumption and a decrease of query processing efficiency.

The topic of this article has been the efficient support of transaction time in the relational model. The concrete results include:

- a transparent extension of the relational model (DM/T) where the transparency is supported by the underlying implementation (IM/T)
- a general query optimization and processing architecture which utilizes partitioned backlog storage, selective pointer and data-view caching, eager/lazy view update, cache indexing, and state transition networks with dynamic programming.
- integration of recomputation and differential computation of queries
- a symmetrical, general notion of differential computation integrating incremental and decremental computation
- formulas for differential computation
- a generalization of the notion of query subsumption to utilize differential computation
- augmentation of standard query optimization with rules for optimization of differential query processing

Several aspects of the individual components of IM/T are the subjects of future research. They include:

- the relative merits of data and pointer caching
- the extension of existing algorithms for the logical access path to the ELAP
- the adaption of existing cache management strategies
- the relative merits of eager and lazy cache update
- the efficient application of state transition networks for query plan enumeration of transaction time queries

Substantial research efforts are required in order to clarify each of these aspects (Snodgrass, 1990). Other research topics include the caching of differential files, statistics for query optimization, optimal algorithms for operators of STNs, and support for general versioning.

References

Ahn, I. Performance Modeling and access methods for temporal database management systems. TR86–018, University of North Carolina, 1986.

Bassiouni, M.A. Data compression in scientific and statistical databases. *IEEE TSE,* 11(10):1047–1058, 1985.

Blakeley, J.A., Coburn, N., and Larson, P. Updating derived relations: Detecting irrelevant and autonomously computable updates. TR, CS–86–17, Computer Science Department, University of Waterloo, Canada, 1986.

Blakeley, J.A., Coburn, N., and Larson, P. Updating derived relations: Detecting irrelevant and autonomously computable updates. *ACM TODS*, 14(3):369–400, 1989.

Bolour, A., Anderson, T.L., Dekeyser, L.J., and Wong, H.K.T. The role of time in information processing: A survey. *ACM SIGMOD Record*, 12(3):27–50, 1982.

Bubenko, jr, J.A. The temporal dimension in information modeling. In: Nijssen, G.M., ed. *Architecture and Models in Data Base Management Systems,* North Holland: Amsterdam, 1977.

Chakravarthy, U.S. and Minker, J. Multiple query processing in deductive databases using query graphs. *Twelfth International Conference on VLDB*, Kyoto, Japan, 1986.

Christodoulakis, S. Analysis of retrieval performance for records and objects using optical disk technology. *ACM TODS*, 12(2):137–169, 1987.

Codd, E.F. A relational model of data for large shared data banks. *CACM* 13(6):377–387, 1970.

Codd, E.F. Extending the database relational model to capture more meaning. *ACM TODS*, 4(4):397–434, 1979.

Dadam, P., Lum, V., and Werner, H.D. Integration of time versions into a relational database system. *Tenth International Conference on VLDB*, Singapore, 1984.

Gunadhi, H. and Segev, A. A framework for query optimization in temporal databases. *Proceedings of the Fifth International Conference on Statistical and Scientific Database Management*, 1989.

Gunadhi, H., Segev, A. and Shantikumar, G.J. Selectivity estimation in temporal databases. TR, LBL–27435, Information and Computer Science Division, Lawrence Berkeley Laboratory, 1989.

Hanson, E.N. A performance analysis of view materialization strategies. *International Conference on the Management of Data*, San Francisco, 1987.

Hong, W. and Wong, E. Multiple query optimization through state transition and decomposition. Memorandum, UCB/ERL M89/25, Electrical Research Lab, College of Engineering, University of California, Berkeley, 1989.

Jarke, M. and Koch, J. Query optimization in database systems. *Computer Surveys*, 16(2):111–152, 1984.

Jarke, M. Common subexpression isolation in multiple query optimization. In: Kim, W., Reiner, D.S., and Batory, D.S., eds., *Query Processing in Database Systems*, Springer-Verlag: Boston, 1984, pp. 191–205.

Jensen, C.S. and Mark, L. A framework for vacuuming temporal databases. CS-TR-2516 and UMIACS-TR-90-105, Department of Computer Science, University of Maryland: College Park, 1990.

Jensen, C.S. and Mark, L. Queries on change in an extended relational model. *IEEE Transactions on Knowledge and Data Engineering*, 4(2):192–200, 1992.

Jensen, C.S., Mark, Leo and Roussopoulos, N. Incremental implementation model for relational databases with transaction time. *IEEE Transactions on Knowledge and Data Engineering*, 3(4):461–473, 1991.

Jhingran, A. A performance study of query optimization algorithms on a database system supporting procedures. *Fourteenth International Conference on VLDB*, Los Angeles, 1988.

Jhingran, A. and Stonebraker, M. Alternatives in complex object representation: A performance perspective. Memorandum, UCB/ERL M89/18 Electrical Research Lab, College of Engineering, University of California, Berkeley, 1989.

Kinsley, K.C. and Driscoll, J.R. Dynamic derived relations within the RAQUEL II DBMS. *ACM Annual Conference*, Detroit, MI, 1979.

Kinsley, K.C. and Driscoll, J.R. A generalized method for maintaining views. *National Computer Conference,* 1984.

Kim, W. Global optimization of relational queries: A first step. In: Kim, W., Reiner, D.S., and Batory, D.S., eds., *Query Processing in Database Systems,* Springer-Verlag: Berlin, 1984, pp. 206–216.

Kolovson, C. and Stonebraker, M. Indexing techniques for historical databases. *Proceedings of the Fifth International Conference on Data Engineering,* 1989.

Lafortune, S. and Wong, E. A state transition model for distributed query processing. *ACM TODS,* 11(3):294–322, 1986.

Lum, V., Dadam, R., Erbe, R., Guenauer, J., Pistor, P., Walch, G., Werner, H., and Woodfill, J. Designing DBMS support for the temporal dimension. *ACM SIGMOD International Conference on the Management of Data,* Boston, MA, 1984.

Mahanti, A. and Bagchi, A. AND/OR graph heuristic search methods. *JACM,* 32(1):28–51, 1985.

McKenzie, L.E. An algebraic language for query and update of temporal databases. TR88–050, Department of Computer Science, University of North Carolina, 1988.

McKenzie, L.E. and Snodgrass, R. An evaluation of algebras incorporating the time dimension in databases. *Computer Surveys,* 23(4):501–543, 1991.

Rich, E. *Artificial Intelligence.* McGraw-Hill: New York, 1983.

Rotem, D. and Segev, A. Physical organization of temporal data. *Proceedings of the Third International Conference on Data Engineering,* 1987.

Roussopoulos, N. View indexing in relational databases. *ACM TODS,* 7(2):258–290, 1982*a*.

Roussopoulos, N. The logical access path schema of a database. *IEEE TSE,* 8(6):563–573, 1982*b*.

Roussopoulos, N. Overview of ADMS: A high performance database management system. *Proceedings of the Fall Joint Computer Conference,* 1987.

Roussopoulos, N. An incremental access method for ViewCache: Concept, algorithms, and cost analysis. *ACM TODS,* 16(3):535–563, 1991.

Roussopoulos, N. and Kang, H. Principles and techniques in the design of *ADMS*±. *Computer,* 19(12):19–25, 1986.

Rowe, L.A. and Stonebraker, M.R., eds. *The Postgres Papers.* Memorandum, UCB/ERL M86/85, Electrical Research Lab, College of Engineering, University of California at Berkeley, 1987.

Salzberg, B.J. and Lomet, D. Access methods for multiversion data. *ACM SIGMOD International Conference on the Management of Data,* Portland, OR, 1989.

Sedgewick, R. *Algorithms,* 2nd edition. Addison-Wesley: Reading, MA, 1988.

Segev, A. and Fang, W. Optimal update policies for distributed materialized views. TR–LBL–26104, Information and Computer Science Division, Lawrence Berkeley Laborarory, 1989.

Segev, A. and Gunadhi, H. Event-join optimization in temporal relational databases. TR–LBL–26600, Information and Computer Science Division, Lawrence Berkeley Laboratory, 1989. Also in *Fifteenth International Conference on VLDB*, Amsterdam, 1989.

Segev, A. and Fang, W. Currency-based updates to distributed materialized views. TR–LBL–27359, Information and Computer Science Division, Lawrence Berkeley Laboratory, 1989. Also in *Sixth International Conference on Data Engineering*, 1990.

Selinger, P.G., Astrahan, M.M., Chamberlain, D.D., Lorie, R.A., and Price, T.G. Access path selection in a relational database management system. *ACM SIGMOD International Conference on the Management of Data*, 1979.

Sellis, T. Global query optimization. *ACM SIGMOD International Conference on the Management of Data*, Washington, DC, 1986.

Sellis, T. Efficiently supporting procedures in relational database systems. *ACM SIGMOD International Conference on the Management of Data*, San Francisco, 1987.

Sellis, T. Intelligent caching and indexing techniques for relational database system. *Information Systems*, 13(2):175–185, 1988*a*.

Sellis, T. Multiple-query optimization. *ACM TODS* 13(1): 23–52, 1988*b*.

Sellis, T. and Shapiro, L. Optimization of extended database query languages. *ACM SIGMOD International Conference on the Management of Data*, Austin, TX, 1985.

Sellis, T., Roussopoulos, N., and Ng, R.T. Efficient compilation of large rule bases using logical access paths. *Information Systems*, 15(1): 73–84, 1990.

Shapiro, L.D. Join processing in database systems with large main memories. *ACM TODS*, 11(3):239–264, 1986.

Shoshani, A. and Kawagoe, K. Temporal data management. *Twelfth International Conference on VLDB*, Kyoto, Japan, 1986.

Smith, J.M. and Chang, P.Y-T. Optimizing the performance of a relational algebra interface. *CACM*, 18(10):569–579, 1975.

Snodgrass, R. The temporal query language TQuel. *ACM TODS*, 12(2):247–298, 1987.

Snodgrass, R. Temporal databases: Status and research directions. *ACM SIGMOD Record*, 19(4):83–89, 1990.

Snodgrass, R. and Ahn, I. A taxonomy of time in databases. *ACM SIGMOD International Conference on the Management of Data*, Austin, TX, 1985.

Snodgrass, R. and Ahn, I. Partitioned storage for temporal databases. *Information Systems*, 13(4):369–391, 1988.

Stam, R.B. and Snodgrass, R. A bibliography on temporal databases. *Data Engineering*, 7(4):53–61, 1988.

Stamenas, A.G. High performance incremental relational databases. UMIACS–TR–89–49, CS–TR–2245, Department of Computer Science, University of Maryland, 1989.

Ullman, J.D. *Principles of Database Systems,* 2nd edition, Computer Science Press: Rockville, MD, 1982.

Wong, E. and Youseffi, K. Decomposition—A strategy for query processing. *ACM TODS*, 1(3):223–241, 1976.

Yang, H.Z. and Larson, P.-Å. Computing queries from derived relations. *Eleventh International Conference on VLDB*, Stockholm, 1985.