Concise Papers

Queries on Change in an Extended Relational Model

Christian S. Jensen and Leo Mark

Abstract— **Change is often an important aspect in database system applications. We provide a data model that allows for the storage of detailed change history in so-called backlog relations. Its extended relational algebra, in conjunction with the extended data structures, provides a powerful tool for the retrieval of patterns and exceptions in change history. We introduce an operator, Σ, based on the notion of compact active domain. It groups data, not in predefined groups, but in groups that fit the data. This novel operator further expands the retrieval capabilities of the algebra. The expressive power of the algebra is demonstrated by examples, some of which show how patterns and exceptions in change history can be detected. Sample applications of this work are statistic and scientific databases, monitoring (of databases, manufacturing plants, power plants, etc.), CAD, and CASE.**

Index Terms— **Change history, historical queries, relational algebra, relational model, retrieval of exceptions, retrieval of patterns, rollback databases, statistical queries, temporal databases, transaction time.**

I. Introduction

Shortcomings of current commercially available database products have been recognized for some time. For instance Jackson [13] writes in the introduction of the Jackson Systems Development (JSD) method:

> "A database is essentially a snapshot; it captures a single state of the reality it models, just as a photograph captures a single state of its subject at a single moment in time. [. . .] A fundamental principle of JSD is that a dynamic real world cannot be modeled by a database."

In order to support retrieval of change history, we need the capability to store not only the most recent snapshot, but the changes that lead to it as well.

DM/T (Data Model with Time) is an extension of the basic relational model that supports transaction time. Using its extended relational algebra, one can retrieve information about previous database states and about normal and exceptional change behavior.

An extension or generalization of functionality should always be augmented with proper default behavior, making the extension transparent to users that do not exploit it. DM/T obeys this principle. We remain within first normal form, in that the relational algebra operators of the extended query language, except for the time-slice operator, manipulate only standard, "flat" relations where the time dimension is eliminated. Thus, the standard relational operators keep their standard semantics. In this sense, the model is a *minimal* extension of the relational model. In part, the transparency is achieved by introducing a separate, system generated and maintained relation, a backlog, for each user defined relation. The backlog contains the complete change history of its associated relation. To our knowledge, we are the first to use this approach in temporal databases. Previously, history information has been put in the relation itself. The backlog approach allows for convenient storage of more historical information than does traditional approaches, thus making it possible to store and conveniently retrieve requests for insertions, modifications, and deletions.

In order to facilitate retrieval of patterns and exceptions related to the change history, an extended query language is required. The operator Σ is the central extension provided by the relational algebra of DM/T. Based on the notion of compactness of active domains, this operator groups under given restrictions the elements of a domain in a relation so that the elements of each group are contiguous. Several results can be returned by Σ: the tuples of all groups generated, tuples of single groups, cardinalities of groups, and ranges of groups. The extension also includes an *aggregate formation* operator (ξ), a *unit* operator (Υ), a *fold* and an *unfold* operator (ϕ and ϕ^{-1}, respectively), and a *when* operator (ω).

The backlogs of DM/T, paired with the standard relational algebra, provides many new possibilities for formulating queries on change history. When the algebra is extended with the operators ξ, Υ, ϕ, ϕ^{-1}, Ω, the expressive power increases, and when the operator Σ is added, the possibilities are further expanded.

We neither address the issue of integrity constraints nor efficient implementation of DM/T in this paper. An implementation model for the relational model extended with transaction time is presented in [14]. This model exploits techniques of differential (incremental and decremental) computation in the context of deferred update of persistent views and is a natural generalization of the work of Roussopoulos [23]–[25].

The contents of this paper are related to issues of temporal databases where numerous temporally oriented extensions of the relational model have already been proposed, e.g., [34], [8], [4], [31], [2], [35], [5], [6], [20], [33]. While we support transaction time, the vast majority of the work has focused on logical time. Our approach has been to make a simple, transparent, first normal form extension which contrasts the previous ones that generally are elaborate NF^2 extensions. The contributions most closely related to ours are a relational algebra that supports transaction time [19] and the relational algebra extended to support logical time while remaining in first normal form [18].

The operator Σ is based on the notion of compact domain, also used in scientific and statistical databases [11]. While compactness has been used as the basis for the definition of statistical normal forms [11], we use a generalized form of compactness as the basis for the definition of a relational algebra operator. In statistical and scientific databases, the focus has been on sampling, nearest neighbor search, estimation and interpolation, transposition and summary operators; and efforts have been put into creation and manipulation of summary tables [22], [30], [28], [10]. The operator Σ presents an attempt to group data according to the data themselves, not according to predefined intervals, in order to retrieve information about normal

Manuscript received September 13, 1989; revised April 30, 1990. The work was carried out at the University of Maryland and was supported by the National Science Foundation under Grants IRI-8719458 and AFOSR-89-0303. C. S. Jensen was also supported by Aalborg University and L. Mark was also supported by the University of Maryland Institute for Advanced Computer Studies and Systems Research Center.

C. S. Jensen was with the Department of Computer Science, University of Maryland, College Park, MD. He is now with the Department of Mathematics and Computer Science, Institute for Electronic Systems, Aalborg University, DK-9220 Aalborg {0}, Denmark.

L. Mark is with the Department of Computer Science, University of Maryland, College Park, MD 20742.

IEEE Log Number 9106263.

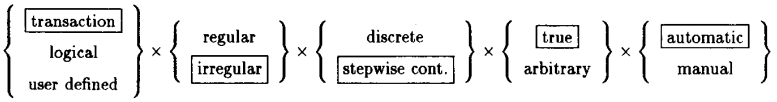

Fig. 1. The figure outlines the characteristics of the time concept of DM/T.

and exceptional change behavior. In [36], a statistical interface for historical relational databases is presented. A so-called enumeration operator is added to the relational algebra of a data model supporting logical time. This operator can be used for generating meaningful summary data in the presence of the time dimension. However, it does not resemble the Σ operator. We have not found a similar operator, and there have been no attempts to combine a transaction time extension and advanced statistical operators into a single data model.

The rest of the paper is structured as follows. Section II, "Time in the DM/T Data Model," describes the time concept supported by DM/T. Section III, "Data Structures of the DM/T Data Model," presents the different types of relations in DM/T with special focus on the backlog relation. Section IV, "Query Language of the DM/T Data Model," presents the basic query language, consisting of the operators of the standard relational model and the *aggregate formation*, *unit*, *fold/unfold*, and *when* operators. Section V, "Query Language Extension based on Compactness," introduces the Σ operator, based on the concept of compact domain, which allows for retrieval of patterns and exceptions. The underlying conceptual framework is presented, the operator is defined, and the expressive power of the operator is illustrated by sample queries. The last section is "Conclusion and Future Research."

II. Time in the DM/T Data Model

The topic of this section is the time concept that we have chosen to support in DM/T. Its characteristics are outlined in Fig. 1. For a more elaborate discussion of time concepts, see [29].

To understand the distinction between transaction time and logical time, observe that a database models a part of reality and is itself a part of (a different part of) reality. Transaction time is the time when facts are entered into the database; it belongs to the world of the database system, and it is independent of the part of reality modeled. In contrast, logical time is the time when facts took place in the modeled reality, and it is independent of when this is registered in the database. User-defined time is merely an uninterpreted domain (e.g., string, integer, and real). DM/T supports transaction time, and its databases are termed *(static) rollback databases* [19], [32].

A time domain can be either *regular* or *irregular*. It is said to be regular if the distances between consecutive values of the active domain are identical. Otherwise, it is irregular. Our transaction time domain is irregular; transactions updating relations occur irregularly.

A time domain can be either *discrete* or *stepwise continuous*.[1] Facts with discrete time stamps are valid only at the exact time of their time stamps. In contrast, in a stepwise continuous domain, facts have an interval of validity. Our time domain has this property because, until a relation is changed by another transaction, the data as they were after the previous transaction are valid, i.e., they are part of the current state of the database.

We use *true* time as opposed to *arbitrary* time. True time reflects the actual time. Thus, we assume the existence of a system clock correctly reflecting the real time. A domain characterized by arbitrary time needs only to have a total order and a metric. A simple count mechanism would be sufficient to support such a domain.

Time stamping can be done *automatically* and *manually*. We use automatic time stamping, which is a natural choice for recording transaction times. Manual time stamping is a natural choice for supplying logical time stamps.

The minimal *time unit* is arbitrarily chosen to be seconds. Thus, we do not provide limitless precision. It is possible to control the granularity using a *unit* operator, Υ. The default unit is minutes (At the implementation level, time stamps are unique.)

Finally, there still remains the question as to *what to attribute with time*, i.e., where time enters our data model. The two relevant alternatives are attribute value stamping and tuple stamping. Each alternative has its advantages and disadvantages [5]. We have chosen the latter because it allows DM/T to remain within first normal form.

III. Data Structures of the DM/T Data Model

In this section, we present the different kinds of relations in the model, i.e., backlog, base relation, view, and backlog view.

A. Backlogs

A *backlog*, B_R, for a relation, R, is a relation that contains the complete history of change requests to relation R [26]. Let the schema of a relation R be $R(A_1 : D_1, A_2 : D_2, \cdots, A_n : D_n)$. The corresponding, system generated and maintained backlog, B_R, is defined as $B_R(Id : SURROGATE, Op : \{Ins, Del, Mod\}, Time : TTIME, A_1 : D_1, A_2 : D_2, \cdots, A_n : D_n)$.

Each tuple in a backlog is a *change request*. Backlog B_R contains three attributes in addition to the attributes of R. *Id* is defined over a domain of surrogates, i.e., logical, system generated unique identifiers. Surrogate values can neither be changed nor seen by users/application programs, but they can be referenced [12]. Each *Id* represents a change request. The attribute *Op* is defined over the enumerated domain of operation types, and values of *Op* indicate whether an insertion (*Ins*), a deletion (*Del*), or a modification (*Mod*) is requested. Finally, the attribute *Time* is defined over the domain of transaction time stamps, *TTIME*, described in the previous section.

The effects on backlogs resulting from update requests on their corresponding relations are described in Fig. 2. Change requests that request the deletion or modification of a nonexisting tuple and change requests that request the insertion of an existing tuple are illegal. It is assumed that the implementation of DM/T rejects such requests.

Physically, backlog relations of tuples of change requests are stored as files of records. Insertion and deletion requests are stored as insertion and deletion records, and modification requests are stored as ordered pairs of a deletion and an insertion record, both with the same time stamp (to distinguish them from unrelated deletion and insertion records).

Example: We introduce a sample relation *Mac* with information about machines: their name, id, size, temperature, material consumption, and yield. The relation has this schema: $Mac(Name : STRING, Id : SURROGATE, Size : \{Sm, Med, Lg\}, Temp : INT, Cons : REAL, Yield : REAL)$. If we use the notation "Tuple: *Mac*" as a shorthand for the attribute definitions

[1] We talk about virtual, simulated continuity.

The Effect on Backlogs of Update Requests	
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*(R,key,-))
modify R(key, new values)	insert B_R(id, *Mod*, time, *tuple*(R,key,new values))

Fig. 2 · The table defines system controlled insertions into backlogs. The function *tuple* takes as arguments a relation name, valid key information for the relation, and an optional list of changes to the identified tuple. It returns the identified and possibly updated tuple.

B_{Mac}								
Id	*Op*	*Time*	*Mac.id*	*Name*	*Size*	*Temp*	*Cons*	*Yield*
1	Ins	Jan. 1, 1991 11:31 a.m.	α	A	Med	73	0	02
2	Ins	Jan. 1, 1991 1:09 p.m.	β	B	Sm	77	0	0
3	Ins	Jan. 1, 1991 1:34 p.m.	γ	C	Lg	78	0	0
4	Mod	Jan. 7, 1991 8:00 a.m.	α	A	Med	343	3.4	70
5	Mod	Jan. 7, 1991 8:04 a.m.	β	B	Sm	386	2.1	84
6	Mod	Jan. 7, 1991 9:00 a.m.	α	A	Med	361	2.9	74
7	Mod	Jan. 7, 1991 9:46 a.m.	α	A	Med	358	3.0	75
8	Mod	Jan. 7, 1991 10:39 a.m.	α	A	Med	359	3.1	73

Fig. 3. The figure illustrates a sample extension of backlog relation B_{Mac} with *NOW* = Jan. 7, 1991 11:00 a.m.

Mac(*NOW* − *90 minutes*)					
Id	*Name*	*Size*	*Temp*	*Cons*	*Yield*
α	A	Med	361	2.9	74
β	B	Sm	386	2.1	84
γ	C	Lg	78	0	0

Mac					
Id	*Name*	*Size*	*Temp*	*Cons*	*Yield*
α	A	Med	359	3.1	73
β	B	Sm	386	2.1	84
γ	C	Lg	78	0	0

Fig. 4. Time-slices of relation *Mac*, where *NOW* = Jan. 7, 1991 11:00 a.m.

of the relation *Mac*, then the backlog of *Mac* has this schema: $B_{Mac}(Id : SURROGATE, Op : \{Ins, Del, Mod\}, Time : TTIME, Tuple : Mac)$. A sample extension of B_{Mac} is given in Fig. 3. □

B. Base Relations and Views

As a consequence of the introduction of time stamps, a base relation is now a function of time. To retrieve a base relation, it must first be time-sliced. Let R be any base relation, then the following are examples of *time-slices* of R:

$$R(t_{init}) \stackrel{\text{def}}{=} R_{init}$$
$$R(t_x) \stackrel{\text{def}}{=} R \text{ "at time } t_x\text{"}, t_x \geq t_{init}$$
$$R \stackrel{\text{def}}{=} R(NOW).$$

When the database is initialized, it has no history, and it is in an initial state, possibly with every relation equal to the empty set. If R is parameterized with an expression that evaluates to a time value, the result is the state of R as it was at that point in time. It has no meaning to use a time from before the database was initialized and after the present time. If R is used without any parameters, this indicates that the wanted relation is the current R. We also introduce the special variable NOW which assumes the time when the query is executed.

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

Example: Fig. 4 shows the extension of *Mac* at two points in time. See also Fig. 3. □

A view is time dependent if at least one of the relations and views, from which it is derived, is time dependent; otherwise, it it fixed. A traditional view is derived from time-sliced base relations. A view derived directly from at least one backlog, i.e., not via a time-sliced base relation, is a backlog view. Backlog views are time-sliced as are base relations and views:

$$B_R(t_x) \stackrel{\text{def}}{=} \sigma_{Time \leq t_x} B_R$$
$$B_R \stackrel{\text{def}}{=} B_R(NOW).$$

Backlog view time-slices involving *NOW* are time dependent, and so are backlog views derived from views involving *NOW*.

C. Overview of Data Structures

We have introduced six types of derived relations:

$$\left\{\begin{array}{c}\text{base relation}\\ \text{view}\\ \text{backlog view}\end{array}\right\} \times \left\{\begin{array}{c}\text{time dependent}\\ \text{fixed}\end{array}\right\}.$$

We distinguish between *base relations*, traditional *views*, and *backlog views*. The only difference between base relations and views is that base relations are derived directly from backlogs while views are derived indirectly, via base relations. A view is valid only at a single point in time, namely the time specified when it was produced, using the query language. A backlog has an associated lifespan. It ranges from the time when the corresponding base relation was created until the current time, if the backlog still exists, or until the backlog was deleted. Backlog views inherit this notion of lifespan.

The second dimension distinguishes between *fixed* and *time dependent*. The valid time of fixed time-slices of base relations and views and the lifespan of fixed backlog views never change. Because it is possible to use the special variable *NOW* in query expressions, both base relations, views and backlog views can be time dependent. A time dependent base relation can be visualized as a view sliding along a backlog as time passes. Similarly, a backlog view can be thought of as a window where one or both ends (start time and end time) move along a backlog.

IV. Query Language of the DM/T Data Model

To take full advantage of the time extension and the additional data structures, we need an extended query language. We use the standard relational algebra as a basis for such an extension. In Section IV-A, we present and briefly discuss the operators of our data model. In Section IV-B, we illustrate the utility of parts of the query language.

A. Operators and Notation

We have already introduced the time-slice operator. Because any relation must be time-sliced before it is manipulated by other operators, the time dimension is eliminated, and standard relational operators can be used. We include the fundamental operators: projection (π), selection (σ), Cartesian product ($\times$), difference ($-$), and union ($\cup$). For projection, we will use the shorthand $\pi_{Tuple} B_R$ to mean projection on all attributes of R. Temporal expressions, i.e., expressions that evaluate to a value of domain $TTIME$ (or a derived domain, see the *unit* operator below), can be used in selection criteria. We will also feel free to use derived operators such as natural join ($\bowtie$), semi-join ($\rhd$), and intersection ($\cap$).

Conceptually, time-slices have transaction time attributes. In order to comply with the *transparency principle*, the transaction time attribute in time-slices of user-defined relations is not displayed. A transaction time attribute can be displayed by means of an explicit projection. In a backlog relation, the time stamp attribute is displayed, and a projection is required to remove it.

We allow for a full range of aggregate functions: *max*, *min*, *mean*, *count*, *avg*, *sum*, and *product*.

To accommodate the aggregate functions, we introduce a generalization, ξ, of the aggregate formation operator [17], [1]. The operator ξ is used for application of an aggregate function to sets of values of an attribute. We use the notation $\xi_{X,\ att\_name=agg\_fct}\ R$ where X is a grouping specification, *att_name* is a new attribute name, and *agg_fct* is an aggregate function. The result of the query is computed the following way: first, the tuples of R are partitioned into the groups implied by X; second, *agg_fct* is applied to each group, and the resulting value is associated with each tuple in the group as a value of the attribute, *att_name*. For example, the following query would return the current average temperature for each machine size:

$$\pi_{Size, Avg\_temp}\ \xi_{Size, Avg\_temp=avg(Temp)}\ Mac.$$

Avg_temp is the new attribute to be generated. One group is generated for each distinct value of *Size*. In general, X can be any sequence of distinct attributes in relation R. In addition, the keyword *Interval* can be substituted for X, meaning that the intervals previously generated by the operator Σ (see the next section) are the groups to be used.

We extend the query language with the *unit* operator, Υ. It is used for changing units and precision of attribute domains. We focus on the precision aspect of the operator and we will not discuss the important aspects of conversion of compatible units. From the literature [16], [9], it follows that such conversions can be done algorithmically, applying simple linear algebra techniques. Also, the issue of information loss due to finite arithmetic is beyond the scope of this presentation. The syntax is as follows:

$$\Upsilon_{A_{i1}=D'_{i1},\ A_{i2}=D'_{i2},\ \cdots,\ A_{ik}=D'_{ik}}$$
$$R(A_1 : D_1, A_2 : D_2, \cdots, A_n : D_n)$$

where $1 \le i_j \le n$, $j = 1, 2, \cdots, k$. The result of this query is relation R with domain D_{i_j} of attribute A_{i_j} changed to D'_{i_j}, $1 \le j \le k$.

Units			
$Domain_1$	$Domain_2$	*Factor*	*Decimals*
D_{i_1}	D'_{i_1}	k_{i_1}	c_{i_1}
D_{i_2}	D'_{i_2}	k_{i_2}	c_{i_2}
	...		
D_{i_k}	D'_{i_k}	k_{i_k}	c_{i_k}

Fig. 5. The relation *Units*.

A special relation, *Units*, contains information about conversions between domains. It has four attributes: $Domain_1$ and $Domain_2$ are character strings listing domain names; *Factor* is a real number, and *Decimals* is an integer. See Fig. 5.

The jth entry in relation *Units* tells that an element in domain D_{i_j} can be transformed into an element in domain D'_{i_j} by multiplying it with k_{i_j} and, using scientific notation, allowing c_{i_j} decimals. The user can insert, delete and modify tuples from this relation. In accordance with the normal convention, the operator rounds off to the closest value.

We have chosen the default unit of the domain *TTIME* to be minutes and the lowest unit to be seconds. Tuples, allowing for *Second*, *Minute*, *Hour*, *Day*, *Week*, *Month*, and *Year*, have been inserted into *Units*.[2] When possible, without causing ambiguity, we do not distinguish between the domain *TTIME* and its compatible domains.

We stamp change request tuples with time values that indicate when the requests were issued. In addition, it sometimes is convenient to know when a requested change is superseded by a more recent change request and is no longer effective. Thus, we integrate a *fold* operator (o) and an inverse *unfold* operator (o^{-1}) [18]. The operator, ϕ, can be used on an unfolded relation,[3] and it transforms the attribute defined over the domain *TTIME* into two attributes, *From* and *To*, both defined over domain *TTIME*. Used on an already folded relation, it produces the identity. Similarly, ϕ^{-1} produces the identity when applied to an already unfolded relation. Fig. 6 shows a sample query and its result. See also Fig. 3.

First, the query in Fig. 6 time-slices *Mac*, then attributes are projected—note that *Time* is projected. Finally, the backlog is used to expand the attribute *Time* of the qualifying tuples into attributes *From* and *To*. Note that an interval of validity of a change request may expand beyond the time used in the time-slice and up to *NOW*.

The next extension is the time related operator, *when* (Ω) [5]. It is used for retrieval from backlogs of times when a specified condition became true (unfolded argument) or was true (folded argument). The following query returns as result Jan. 7, 1991 8:00 a.m. (see Fig. 3)

$$\Omega_{Name=A \wedge 300 \le Temp \le 350} B_{Mac}(\textit{Jan. 7, 1991 10:34 a.m.}).$$

The attribute name of the unary relation returned is *From*. If the Ω operator is used on a folded relation, it returns intervals of validity.

[2] For further discussion of units, see [7].

[3] Application of ϕ to other relations than backlogs has been permitted due to its practicality. However, note that B_R is needed to compute ϕR which may be theoretically inappropriate.

The query below returns the interval Jan. 1, 1991 11:31:07 a.m.–Jan. 7, 1991 7:59:39 a.m.

$$\Upsilon_{From=Second,\ To=Second}\ \Omega_{Name=A \wedge Temp \leq 300}$$
$$\phi\ B_{Mac}(\textit{Jan. 7, 1991 10:34 a.m.}).$$

The attribute names of the binary relation returned by Ω are *From* and *To*.

B. Sample Queries

In Fig. 4 of Section III-B, we illustrated how simple retrievals involving base relations are formulated. Here, we present a number of more complicated queries. First, queries on traditional base relations and views are presented. Then, it is discussed how queries on backlogs are helpful in answering queries on change history, and we show some queries using the new operators. *Retrievals involving base relations and views:*

$$Run\_Mac(t) = \sigma_{Cons>0}\ Mac(t) \tag{1}$$
$$\sigma_{Temp \leq 375}\ Run\_Mac(NOW) \tag{2}$$
$$\sigma_{Size=Med}\ Run\_Mac(NOW). \tag{3}$$

In (1), we define a view. A definition does not result in any computation, all that happens is that the query expression itself is stored in the DBMS. The first step in evaluating any query is to time-slice the constituent relations. So, to retrieve data from the view, an expression that evaluates to a value in the domain *TTIME* must be supplied and substituted for t. Then, the selection is computed. In (2) all currently cool and running machines are retrieved This is a time-dependent time-slice that involves several levels of computation. First, Run_Mac is substituted by its definition. Second, the time-slice is computed. Third, the selections are performed. In the last query (3), all currently running medium machines are retrieved. Note that our descriptions of execution steps merely serve to define the semantics of the queries; optimization may give different sequences or sets of computation steps [15].

Retrievals involving backlogs: Before we turn our attention toward queries directly involving backlog relations, let us look at how the use of backlogs supplement that of usual relations. Suppose we want the changes to *Mac* between t_x and t_y, then these are all plausible candidate queries:

$$Mac(t_y) - Mac(t_x) \tag{4}$$
$$\sigma_{t_x \leq Time \leq t_y} B_{Mac} \tag{5}$$
$$\pi_{Tuple} \sigma_{t_x \leq Time \leq t_y} B_{Mac} \tag{6}$$
$$\pi_{Tuple} \sigma_{t_x \leq Time \leq t_y \wedge (Op=Ins \vee Op=Mod)} B_{Mac}. \tag{7}$$

The result of query (4) is the tuples in *Mac* at t_y, not in *Mac* at t_x. This does not tell us what took place between t_x and t_y. For example, we might not retrieve any deletions that took place in the time interval. If we really want to know what took place between t_x and t_y, we would be better off using the backlog of *Mac*. We have to make clear precisely what we want. Let us look at some possibilities. Query (5) retrieves all possible information about what happened to *Mac*. Insertions, deletions, and modifications are distinguished, and the times when the requests were placed are available. Query (6) eliminates the special backlog attributes from the result. Thus, several changes back and forth between identical *Mac* tuples will be eliminated due to duplicate removal, and it will not be possible to distinguish between operation types anymore. Finally, in (7), we have retrieved all machines that changed, either because they were inserted or because their previous properties were updated.

$\sigma_{Time}\ \pi_{Name,\ Time,\ Temp}\ Mac(\textit{Jan. 7, 1991 8:30 a.m.})$

Name	From	To	Temp
B	Jan. 7, 1991 8:04 a.m.	*NOW*	386
A	Jan. 7, 1991 8:00 a.m.	Jan. 7, 1991 9:00 a.m.	343
C	Jan. 1, 1991 1:34 p.m.	*NOW*	74

Fig. 6. The figure illustrates the retrieval of a folded relation. The query is issued on Jan. 7, 1991 10:00 a.m.

The list of possibilities is by no means exhaustive, but it illustrates how a large number of detailed queries are easily formulated using backlogs.

In query (8), we change the domain of attribute *Time* to *Day* and select all *Del* change requests inserted after Jan. 5, 1991. Then, the time of the first deletion is found and tagged to each of the selected tuples. To find all existing machines, at t_x, that changed properties between t_a and t_b, we issue query (9) which involves both a time-sliced base relation and a backlog. Query (10) results in a list $(Name,\ Time)$ of machines with property changes in the given interval. It is assumed that the tuple $(TTIME,\ Hour,\ 1/60,\ 0)$ is present in *Units*. We achieve a coarser granularity. Thus, if a machine changed more than once within the same hour, it will not be visible in the result.

In summary, we have shown how to conveniently retrieve detailed information about change history of relations. In particular, we have demonstrated the usefulness of backlog relations.

V. Query Language Extension Based on Compactness

This section presents the operator Σ. First, we motivate the need for a query language extension that allows for retrieval of patterns and exceptions from a database. Second, we present a general framework for compactness of active domains. Third, we base the Σ operator on this framework and present a notation for compactness queries. Fourth, we illustrate with sample queries, how to use the operator.

A. Motivation

Databases provide us with the technical ability to store very large amounts of constantly changing data. In databases where the amount of data is larger than users can ever consume, single database states often become less important, whereas the question of whether or not states are reasonably close to what should be expected becomes more important. Similarly, in databases where the rate of change is higher than users can follow, the change itself often becomes less important, whereas the question of whether or not it is reasonably close to what it should be becomes more important.

Powerful abstraction tools must be developed to allow users to identify and subsequently ignore database states that are normal so they can concentrate on unusual and unexpected states. Similarly, powerful abstraction tools must be developed to identify and subsequently ignore changes that reflect normal and expected patterns

Retrievals involving backlogs, views, and new operators:

$$\xi_{-,Min=min(Time)}\ \sigma_{Jan.\ 5,\ 1991 < Time \wedge Op=Del}\ \Upsilon_{Time=Day}\ B_{Mac} \tag{8}$$
$$Mac(t_x) \ltimes \pi_{Tuple}\ \sigma_{t_a \leq Time \leq t_b \wedge Op=Mod}\ B_{Mac} \tag{9}$$
$$\pi_{Name,\ Time}\ \sigma_{Jan.\ 2,\ 1991\ 10\ p.m. \leq Time \leq Jan.\ 6,\ 1991\ 10\ p.m. \wedge Op=Mod}\ \Upsilon_{Time=Hour}\ B_{Mac}. \tag{10}$$

of change, making it possible to concentrate on changes that reflect unusual and unexpected changes of reality.

Our goal is to provide the database with capabilities for identifying relevant and representative states and changes, and exceptional states and changes. Applications of these capabilities include a number of surveillance and monitoring systems.

We will present a framework that opens to a new world of summary and statistics queries that allow for retrieval of change behavior and of common patterns and exceptions in the change behavior.

Queries like the following will be possible.

- Given a unit of measure and an attribute of a relation, group the values into intervals and return the interval which has the smallest cardinality (number of tuples).
- Given a cardinality and an attribute of a relation, group the values into intervals and return all intervals with at least the given cardinality.
- If the given attribute is *Time*, with domain TTIME, of the backlog B_R of a relation R, it will be possible to answer queries like these
  - When were there many insertions into relation R?
  - What is the average number of deletions per time unit from relation R?
  - When were deletions most frequent?

The exact meanings of the sample queries above are not specified. It is the purpose of the next subsections to formalize what they can mean.

B. Compact Domains

Let a relation S be given as $S(C_1, C_2, \cdots, C_p, D_1, D_2, \cdots, D_n)$. We assume that domain names are unique and omit attribute names. We are interested only in finite domains for which there exist a metric and a total order.[4] Let the domains D_i be of this kind and let

$$D_i = \{\alpha_j^i\}_{j=1}^{m_i}, \qquad i = 1, 2, \cdots, n.$$

We drop the top and/or bottom indexes whenever possible without causing ambiguity. Associated with each domain D_i in S is an *active domain*, $D_i(t)$, consisting of the values from D_i present in S at time t. Denote the metric and total order on D_i, $i = 1, \cdots, n$, $dist_i$ and $\geq_i$, respectively. We will feel free to use the derived predicates $\leq$, $=$, $<$, and $>$ where it is convenient.

We want to include both simple domains, as above, and domains composed from the simple domains into our framework. Thus, we have to equip composite domains with both metrics and total orders. Consider the observations:

Observation 5.1: For any composite domain $D_{i_1} \times \cdots \times D_{i_k}$ the induced metric, $dist'$, given below is a metric.

$$dist'((\alpha^{i_1}, \alpha^{i_2}, \cdots, \alpha^{i_k}), (\beta^{i_1}, \beta^{i_2}, \cdots, \beta^{i_k})) = \sqrt{\sum_{j=1}^{k} dist_{i_j}(\alpha^{i_j}, \beta^{i_j})^2}.$$

Observation 5.2: The induced metric $dist''$ defined as one plus the number of elements between two elements if the elements are distinct and zero otherwise, is a metric.

These observations are significant because they tell that if we are given metrics of the simple domains, then induced metrics exist for all composite domains, and because they tell that the user has a *choice* of metric. By construction, the next observation ensures the existence of an order on composite domains given orders on the constituent domains.

Observation 5.3: A composite domain $D_{i_1} \times \cdots \times D_{i_k}$ has a total, lexicographic order $\geq$ defined as follows in terms of the total orders $\geq_{i_j}$ of the domains D_{i_j}, $j = 1, 2, \ldots, k$:

$$\alpha = (\alpha^{i_1}, \alpha^{i_2}, \cdots, \alpha^{i_k}) \quad \beta = (\beta^{i_1}, \beta^{i_2}, \cdots, \beta^{i_k})$$
$$\alpha \geq \beta \overset{\text{def}}{\Leftrightarrow} \alpha^{i_1} >_{i_1} \beta^{i_1}$$
$$\vee (\vee_{l=2}^{k} (\wedge_{j=1}^{l-1} (\alpha^{i_j} =_{i_j} \beta^{i_j}) \wedge \alpha^{i_l} >_{i_l} \beta^{i_l}))$$
$$\vee \wedge_{j=1}^{k} (\alpha^{i_j} =_{i_j} \beta^{i_j}).$$

We are now in a position to define the concept of *compact domain* which informally is a domain without holes. Formally, we have

Definition: Let $\geq$ be a total order on D. The active domain $D(t)$, of domain D in a relation S is *compact* if and only if

$$\forall \alpha, \beta, \gamma \in D : (\alpha, \beta \in D(t) \wedge \alpha \geq \gamma \geq \beta) \Rightarrow \gamma \in D(t).$$

□

Note that compactness is an extensional property of a relation (attribute). Also, observe that D can be a composite domain (Observation 5.3), and finally observe that the domains we consider do not include null values.

Some properties of compactness should be mentioned.

Observation 5.4 Compactness is not preserved under subsequence formation. E.g., for some t, let $D(t) = D_1(t) \times D_2(t) \times \cdots \times D_n(t)$ be compact. Then, for the same t, $D'(t) = D_{i_1}(t) \times D_{i_2}(t) \times \cdots \times D_{i_k}(t)$, $1 \leq i_1 < i_2 < \cdots < i_k \leq n$ is not generally compact. Let us illustrate this observation by means of an example. We define

$$D = \{1, 2, 3, 4, 5\} \times \{1, 2, 3, 4, 5\} \times \{1, 2, 3, 4, 5\} \text{ and}$$
$$D(t) = \{(1, 2, 4), (1, 2, 5), (1, 3, 1), (1, 3, 2)\}.$$

It can be seen that $D(t)$ is compact. Now we project out the second domain to get $D'(t)$ which is not compact because the element $(1, 3)$ is missing

$$D'(t) = \{(1, 4), (1, 5), (1, 1), (1, 2)\}.$$

This same example also demonstrates that compactness is not preserved under permutation (e.g., switch the second and third domain).

We defined an active domain to be compact if its elements are consecutive. We term a set of consecutive elements an *interval*, and we now generalize compactness to *partial compactness* by relaxing the restriction that there be only one interval to instead allowing the existence of a *set* of intervals that fullfil various restrictions.

Definition: The active set of values of a domain D in a relation S, $D(t)$, is a *partially compact domain* with respect to given restrictions if and only if the set of intervals partitioning $D(t)$ satisfy the restrictions.

We consider only *maximal* intervals (i.e., given any two intervals I_i and I_j, then $I_i \cup I_j$ is not an interval). The following types of restrictions are possible:

number of intervals The number of intervals can be restricted. Generally, conjunctions and disjunctions of restrictions of the number of intervals can be specified. Let ι denote this quantity.

size of intervals The size of intervals can be restricted. Here we rely on the existence of a metric for D, see Observations 5.1 and 5.2. Combinations of conjunctions and disjunctions can be specified. Let δ denote this quantity.

cardinality of intervals The number of elements in an interval can be restricted. Again, conjunctions and disjunctions can be specified. Let # denote this quantity.

[4] Compare this distinction between domains with both a metic and an order and domains without to the distinction in statistical and scientific databases between category data (measured data) and summary data (parameter data) [31]. The domains of interest here include all summary data and some category data.

mixed restriction Constraints on the number of intervals, their sizes, and cardinalities can be specified. □

Note that the intervals of a partially compact domain partitions the domain. Also, the requirement that intervals be maximal implies that there must be a hole between any two legal intervals. Partial compactness is a generalization of compactness: when we impose the restriction that the maximum number of allowed intervals in a partially compact domain be one, we have a compact domain. As in the case of compactness, partial compactness of composite domains is not in general preserved under subsequence formation and under permutation of the sequence of composition.

Example: To understand why partial compactness is not preserved under permutation of the aggregation sequence, let $D_1 = \{1, 2, 3\}$, $D_2 = \{a, b, c\}$, and let an interval of $D_1(t) \times D_2(t)$ be given by $\{(2, b), (2, c), (3, a)\}$. Then the set $\{(a, 3), (b, 2), (c, 2)\}$ of elements on the permuted domain is not an interval. □

The definitions of compactness rely heavily on the notion of order. Every simple has an order. The lexicographic order on a composite domain, induced by the orders of its constituent atomic domains, assigns a monotonic decreasing importance to atomic values of the composite domain. The first elements are the most significant, the second ones are the next most significant, etc.

C. Notation for Compactness Queries

We introduce the operator, Σ, for expressing queries based on compactness. The operator takes a relation as an argument and returns a relation, thus preserving the closedness of the extended relational algebra. The general notation is given by

$$\Sigma_{[E][D][O]} R.$$

Some explanation is in order. The first subscript E is an expression consisting of any combination of conjunctions, disjunctions, and negations of restrictions of the variables ι, δ, and #. It is defined as follows

$$\begin{aligned} E &\rightarrow E \wedge E \mid E \vee E \mid \neg E \mid (E) \mid Exp \\ Exp &\rightarrow N \leq V \mid V \leq N \mid V = N \mid N \leq V \leq N \\ N &\rightarrow \cdots \mid -10 \mid -9 \mid \cdots \mid 9 \mid 10 \mid \cdots \\ V &\rightarrow \iota \mid \delta \mid \# \end{aligned}$$

The expression E restricts the process of generating intervals: intervals are only generated if the set of intervals fulfills the specified restrictions.

Example: The E expression below makes the operator Σ generate intervals only if

- the total number of intervals (ι) to be generated is less than or equal to 10;
- the width (δ) of any generated interval is larger than or equal to 8;
- there are at least 1000 elements in each generated intervals (#).

$$\iota \leq 10 \wedge 8 \leq \delta \wedge 1000 \leq \#$$

If any of the restrictions cannot be met, then the empty relation is returned. □

The second subscript, D, specifies which active domain (possibly composite) of the argument relation (R) should be used in the interval generation process. If the schema of R is given by

$$R(A_1 : D_1, A_2 : D_2, \cdots, A_n : D_n),$$

then any sequence $A_{i_1}, A_{i_2}, \cdots, A_{i_k}$ where $1 \leq i_1 < i_2 < \cdots < i_k \leq n$, can be specified, provided a metric and a total order exist for each constituent subdomain. If the cardinality of the active domain is zero, then we have an empty interval, and $\iota = 1$, $\delta = 0$.

The third subscript, O, is used for specifying which part of the computed result is to be returned. The syntax is

$$O \rightarrow X : Y$$

where X specify which of the intervals that fulfill E should be part of the result; Y is a specification of exactly which information about the intervals should be in the result.

$$\begin{aligned} X &\rightarrow X, X \mid A\ B \mid C \\ A &\rightarrow \text{max} \mid \text{avg} \mid \text{min} \\ B &\rightarrow \text{range} \mid \text{card} \\ C &\rightarrow \text{all} \\ Y &\rightarrow Y, Y \mid \text{data} \mid \text{range} \mid \text{card}. \end{aligned}$$

Example: The expression "max range: card" returns only the cardinality of the computed interval(s) with the largest range. The expression "all: data" returns a selection on the argument relation with all the tuples that were placed in the generated intervals. Finally, "avg card: range" returns the range of the interval(s) with average cardinality. □

The schemas of the various types of results that can be returned are listed below. If combinations of the keywords, data, range, card are specified, the schemas are simply aggregated.

DATA: $(A_1 : D_1, A_2 : D_2, \ldots, A_n : D_n)$
RANGE: A_{i_1}.From : D_{i_1}, A_{i_2}.From : $D_{i_2}, \cdots, A_{i_k}$.From : D_{i_k}, A_{i_1}.To : D_{i_1}, A_{i_2}.To : $D_{i_2}, \cdots, A_{i_k}$.To : D_{i_k})
CARD: $(Card : Integer)$

D. Sample Compactness Queries

Now that we have introduced the general notation for compactness queries, let us consider some examples.

The query $\Sigma_{[E][Time][all: data]} B_{Mac}$ retrieves all the tuples of B_{Mac} if the restrictions on ι, #, and δ are met. The following algorithm defines the result:

---

init #, δ, ι boundaries * initialization of restrictions implied by E
init Res * the result relation is initially empty
init i * the number of generated intervals, initially 1
init $\delta[i]$ * the size of the ith interval, initially 0
init $\#[i]$ * the cardinality of the ith interval, initially 0
* a clustering index on the domain $TTIME$ is assumed, so no sorting on Time is required
while unread elements in Time

do pick next element e * tuple of B_{Mac} with the lowest Time value
$\#[i]++$ * one more element in interval i
Res $\leftarrow$ Res $\cup$ {e}
while unread elements in Time, and succ(e) is an immediate successor of e
do $e \leftarrow$ succ(e)
$\#[i]++$
Res $\leftarrow$ Res $\cup$ e
check # boundaries
compute δ[i]
check δ boundaries
if unread elements in Time
then $i++$
check ι boundaries

If any of the checks in the algorithm fail, then B_{Mac} with an empty extension is returned. Note that the algorithm is linear in n, (i.e. $\mathcal{O}(n)$), where n is the cardinality of the attribute, Time (i.e. the active domain).

Now, let us consider some more interesting examples shown at the bottom of this page. Statistical normal forms based on compactness and uniformity are important in statistical applications [11]. A relation is in first statistical normal form (1.SNF) if every atomic noncategory domain is compact [11]. Let us illustrate how we can test whether a relation is in 1.SNF. In *Mac*, this means that the active domains of each of the attributes: *Temp*, *Cons*, and *Yield* must be compact. Query (11) returns the current state of relation *Mac* if *Yield* is compact. Otherwise, it returns the empty relation. Using similar queries for each of the attributes, *Temp* and *Cons*, we can determine whether *Mac* is in 1.SNF. Given a relation in 1.SNF, a natural continuation would be to ask if it is also in 2.SNF. This is the case if every atomic noncategory (see Section V-B) domain is also uniform (i.e., whether the number of times each value occurs is the same). It can be tested using the aggregate formation operator and the aggregate function *count*.

The query (12) retrieves the data of the interval(s) during the last seven days where the least number of changes to relation *Mac* occurred. First, the part of the backlog B_{Mac} for the past seven days is selected. Second, the time unit is changed to *Hour*. Third, the tuples of B_{Mac} are grouped according to the Time attribute, and the elements of the interval with the least number of elements are returned as the final result.

In query (13), intervals are again generated on the basis of the attribute Time of B_{Mac}. Here the ranges of the intervals with large cardinalities (i.e., intervals with more than X elements) are the result.

In the next query (14), we find the time intervals over the lifetime of *Mac* where the most insertions took place. Among all the intervals, only the ranges of the one with largest cardinality and the one with largest range are returned.

In (15), we restrict the process of interval generation. See the first example in Section V-C. If it is possible to generate intervals under the imposed restrictions, the whole selection on B_{Mac} is returned; otherwise, B_{Mac}, with an empty extension, is returned.

In query (16), we are interested in deletions to the database that took place after Jan. 7, 1991 8 a.m. and before Jan 8, 1991 8 a.m. Before the Σ operator is applied, we select the interesting parts of the backlog and change the domain of the attribute Time. Then, if at least four intervals result, the ranges and cardinalities of the generated intervals are returned (the schema is defined in Section V-C); otherwise, the empty relation (same schema) is the result.

The final query (17) returns the average number of deletion requests per hour to the relation *Mac*.

In conclusion, we have illustrated the utility of the Σ operator. The reader may compare the sample queries to the informal questions of Section V-A.

VI. Conclusion and Future Research

In this paper, we have pursued the idea of asking questions about the evolution of a database. In doing so, we have focused on abstracting common patterns and exceptions in the change history.

First, we briefly described how we have added transaction time to the standard relational model, and we gave a precise characterization of the time concept chosen.

Second, we briefly described how we have extended the data structures of the standard relational model to include system generated and maintained backlog relations for recording detailed change history. The backlogs contain requests for insertions, deletions, and modifications to their associated base relations rather than the results of the requests. Thus, the backlogs faithfully reflect the evolution of the database. Using only the standard relational algebra, the extended data structures allowed for historical queries.

$$\Sigma_{[\iota=1][Yield][all:\ data]}\ Mac \tag{11}$$

$$\Sigma_{[\,][Time][min\ card:\ data]}\ \Upsilon_{Time=Hour}\ \sigma_{NOW-7\ days\leq Time}\ B_{Mac} \tag{12}$$

$$\Sigma_{[][Time][\#>X\ :\ range]}\ B_{Mac} \tag{13}$$

$$\Sigma_{[\,][Time][max\ card,\ max\ range:\ range]}\ \sigma_{Op=Ins}\ B_{Mac} \tag{14}$$

$$\Sigma_{[\iota\leq 10\wedge 8\leq\delta\wedge 1000\leq\#][Time][all:\ data]}\ \sigma_{NOW-7 days\leq Time}\ B_{Mac} \tag{15}$$

$$\Sigma_{[4\geq\iota][Time][all:\ range,card]}\ \sigma_{Op=Del\wedge Jan.7,1991\ 8a.m.\leq Time\leq Jan8.1991\ 8a.m.}\ \Upsilon_{Time=Hour}\ B_{Mac} \tag{16}$$

$$\xi_{-,Av=Avg(Su)}\ \pi_{Time,Su}\ \xi_{Time,\ Su=count(Id)}\ \Upsilon_{Time=Hour}\ \sigma_{Op=Del}\ B_{Mac}. \tag{17}$$

Third, we extended the standard relational algebra to include the operators *unit* (Υ), *fold* (ϕ), *unfold* (ϕ^{-1}), *when* (Ω), and *aggregate formation* (ξ). By means of sample queries, we demonstrated the retrieval power of the combination of the extended algebra and the extended data structures.

Fourth, we added the novel operator Σ to the query language. Based on the notion of compactness, the operator provides a new way to group data. Normally, data have been grouped according to predefined groups. The idea of the Σ operator is to let the data themselves determine the groups. Thus, the operator groups consecutive data together, and starts a new group when a hole is detected. It is possible to impose restrictions on the grouping process. The number of allowed groups can be specified; the allowed widths and cardinalities of the groups can be specified. Not only can the grouping process be applied to atomic attributes, it can also be applied to composite domains. Additionally, it is possible to select the result of an application of the operator from among the generated groups. We demonstrated the use of the operator by sample queries. The *unit* operator allows for varying the consecutiveness of data, and the *aggregate formation* operator allows for computing statistics on generated groups. As a whole, the query language used in conjunction with the extended data structures allows for retrieving patterns and exceptions in the change history of the database.

We have not been concerned with implementation. Efficient implementation of the data model DM/T has most recently been the topic of [15].

We have introduced the notion of a Σ operator. How to apply the operator in different settings is an issue of further research. This research can lead to other versions of the operator, better suited for specific applications. Because of the ability to generate groups from the data, the operator is well suited for usage during the process where, for example, a statistician tries to get a feel for the data of a database. Integration of browsing capabilities and the operator, allowing for interactive modification of the subscripts, E and O, is a natural future direction with some resemblance to query generalization [21]. Also, the application of the operator to the statistical normal forms of [11] is an interesting direction. Finally, it would be of interest to investigate the possible application of a Σ-like operator to dependency theoretic problems.

ACKNOWLEDGMENT

The authors wish to thank K. Romanik for supplying valuable comments on an earlier draft of the paper, editor M. C. Fernández-Baizán, and the anonymous referees for their comments which improved the paper.

REFERENCES

[1] R. Agrawal, "Alpha: An extension of relational algebra to express a class of recursive queries," in *Proc.Data Eng. Conf.* , Los Angeles, CA, , Feb. 1987, pp. 1–11.

[2] G. Ariav, "A temporally oriented data model," *ACM TODS*, vol. 11, no. 4, pp. 499–527, Dec. 1986.

[3] A. Bolour, T. L. Anderson, L. J. Dekeyser, and H. K. T. Wong, "The role of time in information processing: A survey," *ACM SIGMOD Rec.*, vol. 12, no. 3, pp. 27–50, Apr. 1982.

[4] J. Clifford and A. Croker, "The historical relational datamodel (HRDM) and algebra based on lifespan," *Data Eng.*, pp. 528–537, Feb. 1987.

[5] J. Clifford and A. U. Tanzel, "On an algebra for historical relational databases: Two views," in *Proc. ACM SIGMOD '85*, 1985, pp. 247–.

[6] J. Clifford and D. S. Warren, "Formal semantics for time in databases," *ACM TODS*, vol. 8, no. 2, pp. 214–254, June 1983.

[7] C. J. Date, "A proposal for adding date and time support to SQL," *Sigmod Rec.*, vol. 17, no. 2, pp. 53–76, June 1988.

[8] K. S. Gadia, "A Homogeneous relational model and query languages for temporal databases," *ACM TODS*, vol. 13, no. 4, 418–448, Dec. 1988.

[9] N. Gehani, "Databases and unit of measure," *IEEE Trans. Software Eng.*, vol. SE-8, pp. 605–611, Nov. 1982.

[10] S. P. Ghosh, "Statistical relational tables for statistical database management," *IEEE Trans. Software Eng.*, vol. SE-12, pp. 1106–1116, Dec. 1986.

[11] ——, "Statistical relational databases: Normal forms," Tech. Rep. RJ-6555, IBM Research Division, Almaden Research Center, San Jose, CA, Nov. 1988.

[12] P. Hall, J. Owlett, and S. J. P. Todd, "Relations and entities," in *Modeling in Data Base Management Systems*, G. M. Nijssen, Ed. Amsterdam, The Netherlands: North-Holland, 1976, pp. 201–220.

[13] M. A. Jackson. *System Development*, Prentice-Hall International Series in Computer Science. Englewood Cliffs, NJ: Prentice-Hall, 1983.

[14] C. S. Jensen, L. Mark, and N. Roussopoulos, "Incremental implementation model for relational databases with transaction time," *IEEE Trans. Knowledge Data Eng.*, vol. 3, pp. 461–473, December 1991.

[15] C. S. Jensen, L. Mark, N. Roussopoulos, and T. Sellis, "Using differential techniques to efficiently support transaction time," to be published.

[16] M. Karr and D. B. Loveman III, "Incorporation of units into programming languages," *Commun. ACM*, vol. 21, no. 5, pp. 385–391, May 1978.

[17] A. Klug, "Equivalence of relational algebra and relational calculus query languages having aggregate functions," *J. ACM*, vol. 29, no. 3, pp. 699–717, July 1982.

[18] N. A. Lorentzos and R. G. Johnson, "Extending relational algebra to manipulate temporal data," *Inform. Syst.*, vol. 13, no. 3, pp. 289–296, 1988.

[19] E. McKenzie and R. Snodgrass, "Extending the relational algebra to support transaction time," in *Proc. ACM SIGMOD '88*, 1988, pp. 467–477.

[20] ——, "An evaluation of algebras incorporating time," Tech. Rep. TR-89–22, Dep. Comput. Sci., Univ. of Arizona, Tucson, AZ 85721, Sept. 1989.

[21] A. Motro, "Query generalization: A technique for handling query failure," in *Proc. First Int. Workshop Expert Database Syst.*, Oct. 1984, pp. 314–325.

[22] G. Özsoyoğlu and Z. M. Özsoyoğlu, "Statistical database query languages," *IEEE Trans. Software Eng.*, vol. SE-11, pp. 1071–1081, Oct. 1985.

[23] N. Roussopoulos, "The logical access path schema of a database," *IEEE Trans. Software Eng.*, vol. 8, pp. 563–573, Nov. 1982.

[24] ——, "View indexing in relational databases," *ACM TODS*, vol. 7, no. 2, pp. 258–290, June 1982.

[25] ——, "The incremental access method of view cache: Concept, algorithms, and cost analysis," Tech. Rep. UMIACS-TR-89–15, CS-TR-2193, Dep. Comput. Sci., Univ. of Maryland, College Park, MD 20742, Feb. 1989.

[26] N. Roussopoulos and H. Kang, "Principles and techniques in the design of ADMS±," *IEEE Comput. Mag.*, vol. 19, no. 12, pp. 19–25, Dec. 1986.

[27] N. L. Sarda, "Extensions to SQL for historical databases," *IEEE Trans. Knowledge Data Eng.*, vol. 2, pp. 220–230, June 1990.

[28] A. Shoshani, "Statistical databases: Characteristics, problems, and some solutions," in *Proc. Eighth VLDB Conf.*, Sept. 1982, pp. 208–222.

[29] A. Shoshani and K. Kawagoe, "Temporal data management," in *Proc. Twelfth VLDB Conf.*, Aug. 1986, pp. 79–88.

[30] A. Shoshani and H. K. T. Wong, "Statistical and scientific database issues," *IEEE Trans. Software Eng.*, vol. SE-11, pp. 1040–1047, Oct. 1985.

[31] R. Snodgrass, "The temporal query language TQuel," *ACM TODS*, vol. 12, no. 2, pp. 247–298, June 1987.

[32] R. Snodgrass and I. Ahn, "A taxonomy of time in databases," in *Proc. ACM SIGMOD '85*, 1985, pp. 236–246.

[33] R. Snodgrass, S. Gomez, and E. McKenzie, "Aggregates in the temporal query language TQuel," Tech. Rep. TR 89–26, Dep. Comput. Sci.. Univ. of Arizona, Tucson, AZ 85 721, Nov. 1989.

[34] A. U. Tansel, E. M. Arkun, and G. Özsoyoğlu, "Time-by-example query language for historical databases," *IEEE Trans. Software Eng.*, vol. 15 pp. 464–478, Apr. 1989.

[35] A. U. Tansel, "Adding time dimension to relational model and extending relational algebra," *Inform. Syst.*, vol. 11, no. 4, pp. 343–355, 1986.

[36] ——, "A statistical interface for historical relational databases," in *Data Eng.*, pp. 538–546, Feb. 1987.