UMIACS–TR–89–80
CS–TR–2299

August, 1989

Queries on Change in an Extended Relational Model†

Christian S. Jensen‡
Department of Computer Science

Leo Mark
Institute for Advanced Computer Studies
Department of Computer Science
University of Maryland
College Park, MD 20742

ABSTRACT

A data model is a means of modeling, communicating about, and managing part of reality. In our understanding one of the most fundamental characteristics of reality is change; whereas change is fundamental, stability is relative and temporary. Change is an often critical aspect of database systems applications; in many applications change itself and previous states are of interest. Change presupposes the concept of time. We provide a data model that allows for the storage of detailed historical data in so-called backlog relations. The query language extends the standard relational algebra to take advantage of the additional data. In particular, we introduce an operator *Sigma* based on the notion of compact active domain. This operator groups data, not in predefined groups, but in groups that "fit" the data. The expressive power of the operator is demonstrated by examples showing how patterns and exceptions in change history can be detected. Sample applications of this work are statistic and scientific databases, monitoring (of production systems, databases, power plants, etc.), CAD and CASE.

† Supported by NSF IRI–8719458, and AFOSR–89–0303.

‡ In addition supported by Aalborg University, Denmark.

1 Introduction

Within systems development/SE the shortcomings of current commercially available database products have been recognized for some time. For instance Jackson [Jac83] writes in the introduction to the JSD systems development 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 is an extension of the basic relational model that supports transaction time thus allowing for storage of historical data. Using its extended relational algebra one can retrieve information about normal and exceptional change behavior. In general, 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 1.NF in that the relational algebra operators of the extended query language, except for the time-slice operator, only manipulate standard "flat" relations. Thus the standard relational operators keep their standard semantics. In these senses the model is a *minimal* extension of the relational model, and it promises an easy and efficient implementation.

The transparency is, in part, 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. 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, even if it turned out that these had no effect.

In order to facilitate retrieval of patterns and exceptions related to the change history an extended query language is required.

The Σ operator is the central extension provided by the relational algebra of DM/T. Based on the notion of compactness of active domains this operator tries, under given restrictions, to group the elements of a domain in a relation so that the elements of each group are contiguous. Several results can be returned: The tuples of all groups generated, tuples of single groups, cardinalities of groups, and ranges of groups.

The extension also includes a *unit* operator (Υ), a *fold*, and an *unfold* operator (ϕ and ϕ^{-1}, respectively), a *when* operator (ω) and an *aggregate formation* operator (ξ).

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

We do not address the issue of integrity constraints of DM/T in this paper. Also, issues of efficient implementation are not part of the subject of this paper. Instead we refer the interested reader to [JMR89], where we present an implementation model for a transaction time extension like the one of this paper. The model presented exploits techniques of incremental computation in the context of deferred update of persistent views, and is a natural generalization of the work of Roussopoulos [Rou82a, Rou82b, Rou89].

The contents of this paper touch upon issues of temporal databases and scientific and statistical databases. Numerous temporally oriented extensions of the relational model have already been proposed, including [TAO89, Gad88, LJ88, MS88, CC87, Sno87, Ari86, UT86, CUT85, CW83]. See these for further references. 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 1.NF extension which contrasts the previous ones that generally are elaborate NF^2 extensions. The contributions most closely related are reported in [MS88] where a relational algebra that supports transaction time is found, and in [LJ88] where the relational algebra is extended to support logical time while remaining in 1.NF.

The Σ operator is based on the notion of compact domain and it is related to scientific and statistical databases (SSD) where a similar notion of compact domain is found in [Gho89]. While [Gho89] uses compactness as the basis of the definition of statistical normal forms, we use a generalized form of compactness as the basis of a relational algebra operator. In SSD focus has been put on sampling, nearest neighbor search, estimation and interpolation, transposition and summary operators, and efforts have been put into creation and manipulation of summary tables [OO85, SW85, Sho82, Gho86]. The Σ operator presents an attempt to group data according to the data themselves and not according to predefined intervals in order to retrieve information about normal and exceptional change behavior. In [UT87] 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. It does not, however, 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. In [Mal86] it is described how to statistically treat the information content of a database. There relations are treated as multivariate frequency distributions; and they can be analysed using methods of multivariate information theory, making it possible to measure statistical interdependence among attributes and the effect of one attribute on other ones. The language of this work is "multivariate information theory" and as such the approach deviates from the one of this paper.

The contents of the remaining sections of the paper are:

Section 2, **Time and Change in Databases**, describes the time concept supported by DM/T and discusses tuple versus attribute value time stamping.

Section 3, **Data Structure Extensions of the DM/T Data Model**, presents the different types of relations in DM/T. Special focus is put on the backlog relation.

Section 4, **Query Language of the DM/T Data Model**, present the basic query language consisting of the operators of the standard relational model and the *unit*, *when*, *aggregate formation* and *fold/unfold* operators.

Section 5, **Query Language Extension based on Compactness and Uniformity**, introduces the Σ operator, based on the concept of compact domain, which allows for retrieval of patterns and exceptions. First the conceptual framework is presented. Then the operator is defined, and finally its expressive power is illustrated by sample queries.

The last section contains the **Conclusion and Future Research**.

2 Time and Change in Databases

Until now we have talked about time as a single concept. Time is, however, more complex. In this section we discuss different concepts of time. The purpose is to convey an understanding of the time concept we have chosen to support in DM/T.

Extensive work already has been done on concepts of time. Curious philosophers have been puzzled and thrilled by it for centuries. Within the area of databases, Clifford notes that "time is something so taken for granted that its exact nature is highly elusive" [CUT85]. While we focus on categories of time-domains we do not intend to present a complete account of all the aspects of time. For coverage of and reference to other aspects of the understanding of time in databases we encourage the interested reader to study the surveys of Snodgrass and Ahn [SA85] and Bolour, Anderson, Dekeyser and Wong [BADW82]. All the kinds of time domains we address are characterized by a *total order* of the elements. For discussions of partially ordered time domains, see [BMP] or [Lam]. Also, we will only discuss *absolute* temporal data. Temporal information such

as "x happened 5 minutes before y" or "x took place after z" is relative. For a discussion of relative temporal information, see [Cha88].

Time in reality is perceived uncountably infinite and continuous. Any representation of time of reality in a database is discrete and finite, due to the finite nature of computers.

Fundamentally, a database models a part of reality and is itself a part of (a different part of) reality. From this it follows that we can have an interest in representation of

- **Time in the part of reality which is modeled in the database.**
- **Time in the part of reality that surrounds the data base, the input subsystem.**

EXAMPLE: Assume we have a database with salaries of employees of an organization. The daily working hours, the earnings, the rules for payment and other financial aspects of the employees are *the part of reality we model.*

The database is placed in an office where it is operated by secretaries. The office with its secretaries is *the part of reality that surrounds the database.* This is termed the *input subsystem* [Jac83].

In the area of systems development, the concepts of *edp-system* and *edp-based system* have been applied for years. The first concept is used to indicate that we only consider the technical system itself while the second indicates both the technical system and its organizational setting. See [Mat81] for an excellent discussion.

The time a worker arrives, say 8:00 am, is time in the modeled reality. The time the database is updated to reflect that the worker has arrived, say 8:30 am, is time in the reality surrounding the model, the input subsystem. □

Different names with slightly different meanings have been used for these time concepts. Representation of time in the modeled reality is referred to as logical time, event time, effective time, state, valid time and start/end time. Representation of time in the input subsystem is referred to as physical time, registration time, data-valid-time-from/to, start/end time and transaction time [SA85]. We adopt the terms *logical time* and *transaction time.*

Both logical and transaction time can be erroneous. Transaction time is inherently independent of the part of reality which is modeled while it indeed is dependent on the computer based systems. Logical time is application dependent. Thus, transaction time and logical time are independent time dimensions.

We have chosen to concentrate on transaction time. A database supporting only transaction time is termed a *(static) rollback database* [MS88]. Using a time-slice operator, it is possible to see the database as it appeared at any time during the past. Such a database records the history of the input subsystem, and only if a well-defined mapping exists between logical and transaction time does it model the history of the modeled reality. Well-defined mappings exist in for example monitoring applications.

A database with logical time, a *historical database*, is a poor foundation for queries on change history. To quote Snodgrass and Ahn [SA85]: "[In a *historical database*] Previous states are *not* retained, so it is not possible to view the database as it was in the past."

A database supporting both transaction time and logical time is a *temporal database*. Here relations can be thought of as four dimensional:

$$\text{transaction time} \times \text{logical time} \times \text{attributes} \times \text{tuples}$$

Before we turn our attention to another, orthogonal characterization of time domains let us mention *user defined time*. In present commercially available database products the primitive types are often very limited (e.g. character strings, real and integer). A database supports user defined time if a time domain is one of the primitive types. The attributes defined over the time domain are uninterpreted by the system. User-defined time is independent of the above time concepts. In summary, so far we have these domain types:

$$\left\{ \begin{array}{c} \boxed{\text{transaction time}} \\ \text{logical time} \\ \text{user defined time} \end{array} \right\}$$

The characterizations of time domains to be discussed next are inspired by [SK86]. 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. Statistical data are occasionally regular, i.e. values are recorded at regular intervals during controlled experiments. Our transaction time domain is irregular; transactions updating relations occur irregularly.

$$\left\{ \begin{array}{c} \boxed{\text{transaction time}} \\ \text{logical time} \\ \text{user defined time} \end{array} \right\} \times \left\{ \begin{array}{c} \text{regular} \\ \boxed{\text{irregular}} \end{array} \right\}$$

A time domain can be either *discrete* or *stepwise continuous*[1]. Facts with discrete time stamps are only valid at the exact times of their time stamps. This is, for instance, the case when temperatures

[1] We talk about virtual, simulated continuity.

in a power plant are measured. 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. Ideally, this should be left unspecified, i.e. left to the interpretation of the user depending on the kind of fact that is time stamped. In order to keep the extension simple we have chosen not to support discrete time.

$$\left\{\begin{array}{c}\boxed{\text{transaction time}}\\ \text{logical time}\\ \text{user defined time}\end{array}\right\} \times \left\{\begin{array}{c}\text{regular}\\ \boxed{\text{irregular}}\end{array}\right\} \times \left\{\begin{array}{c}\text{discrete}\\ \boxed{\text{stepwise continuous}}\end{array}\right\}$$

To complete the conceptual description of our transaction time concept, we note that we use *true* time as opposed to *arbitrary* time. True time reflects the actual time of the input subsystem. Thus we assume the existence of a system clock correctly reflecting actual time. A domain characterized by arbitrary time only needs to have a total order and a metric. A simple count mechanism would be sufficient to support such a domain.

$$\left\{\begin{array}{c}\boxed{\text{transaction time}}\\ \text{logical time}\\ \text{user defined time}\end{array}\right\} \times \left\{\begin{array}{c}\text{regular}\\ \boxed{\text{irregular}}\end{array}\right\} \times \left\{\begin{array}{c}\text{discrete}\\ \boxed{\text{stepwise continuous}}\end{array}\right\} \times \left\{\begin{array}{c}\boxed{\text{true time}}\\ \text{arbitrary time}\end{array}\right\}$$

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.

$$\left\{\begin{array}{c}\boxed{\text{transaction time}}\\ \text{logical time}\\ \text{user defined time}\end{array}\right\} \times \left\{\begin{array}{c}\text{regular}\\ \boxed{\text{irregular}}\end{array}\right\} \times \left\{\begin{array}{c}\text{discrete}\\ \boxed{\text{stepwise continuous}}\end{array}\right\} \times \left\{\begin{array}{c}\boxed{\text{true time}}\\ \text{arbitrary time}\end{array}\right\} \times \left\{\begin{array}{c}\boxed{\text{automatic}}\\ \text{manual}\end{array}\right\}$$

The minimal *time unit* used is arbitrarily chosen to be seconds. Note that this means that there exists an $\epsilon > 0$ so that for any two elements, α, β, of our time domain, $dist(\alpha, \beta) \geq \epsilon$. Thus we do not provide limitless precision. [Gad88] terms this discreteness. It is possible to control the granularity using a *unit* operator, Υ. The default unit is minutes.

Time attributes can be either *implicit* and *explicit*. To illustrate, let us briefly mention the choice we made. In user defined and schema relations we chose implicit time stamps. Thus we do not consider time attributes during duplicate removal. This does not mean that the time stamps

are gone; they can be displayed by an explicit projection. In backlog relations we have explicit time attributes.

There still remains the question as to *what to attribute with time*, i.e. where time enters our data model. The two relevant choices are attribute value stamping or tuple stamping. We have chosen the latter. Let us consider the major arguments.

In the literature the opinions are mixed. Some researchers use tuples as units, e.g. Snodgrass, Ariav, Ben-Zvi, etc. [CUT85]. Other researchers argue for attribute values as units, e.g. Clifford, Tansel, etc. The most prominent argument why attribute values are the proper units is that the attributes of relations generally are not interdependent. Four manifestations of independence have been mentioned. First, attributes might be updated at different rates. Second, some attributes might not change at all. Third, some attributes vary continuously over time while others vary stepwise, etc. Fourth, a change to a tuple is usually a change to an individual attribute, not all the attributes of a tuple. As a fifth manifestation, there might be attributes for which we do not care about the history, and attributes where we do care.

The crucial advantage of the tuple approach is that the model becomes comparably simpler than in the attribute value case. The introduction of time stamps on attribute values would have made our model a NF^2-model [CUT85, Gad88]. In addition, if we use a semantically *irreducible* (fact) design discipline where independent facts are separated the independence argument above does not apply. Indeed, it seems that a tuple approach - with its simplicity - married to an irreducible design discipline is a promising approach.

In summary our transaction time concept has the following characteristics: Tuples are automatically stamped with transaction times. Stamps on backlog tuples are explicit while stamps on user-defined and schema relations are implicit. Once entered, time stamped data never change. Transaction times are relative to the system clock which is assumed to reflect the time of the input subsystem. They are irregular and stepwise constant. The default unit was chosen to be minutes, but seconds and coarser granularities were possible.

3 Data Structure Extensions 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, backlog view.

3.1 Backlogs

A *backlog*, B_R, for a relation, R, is a relation that contains the complete history of change requests to relation R.

The schema of relation R and its corresponding backlog is shown in figure 1.

Relation name	R
Attribute name	*Domain name*
A_1	D_1
A_2	D_2
...	...
A_n	D_n

Relation name	B_R
Attribute name	*Domain name*
Id	*SURROGATE*
Op	{*Ins, Del, Mod*}
Time	*TTIME*
A_1	D_1
A_2	D_2
...	...
A_n	D_n

Figure 1: Schema for the relation R and its backlog, B_R.

Each tuple in a backlog is a *change request.* As shown, B_R contains three attributes in addition to the attributes of R. *Id* is defined over a domain of logical, system generated unique identifiers, i.e. surrogates. The values of *Id* represent the individual change requests, they can be referenced but not read by users/application programs. 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*, as discussed in detail in the previous section.

The database management system (DBMS) automatically generates and maintains a backlog for each base relation (i.e. user defined relations and schema relations). Figure 2 shows the effect on backlogs resulting from operation requests on their corresponding relations. When an insertion into R is requested the tuple to be inserted is entered into B_R. When a deletion is requested key information is entered into the backlog and in the case of modification both key information and new values are inserted into the backlog.

The Effect of Requested Operations on Backlogs	
Requested operation on R:	Effect on B_R:
insert R(tuple)	insert B_R(id, *Ins*, time, tuple)
delete R(k)	insert B_R(id, *Del*, time, tuple(k))
modify R(k, new value)	insert B_R(id, *Mod*, time, tuple(k,new value))

Figure 2: System controlled insertions into a backlog. The function "tuple" returns the tuple identified by its argument.

EXAMPLE: We introduce a sample database to illustrate the concept of backlog and other concepts to be presented in the sequel. The database consists of one user-defined relation, *Emp*, with five attributes. Figure 3 shows the schema of *Emp* and its backlog, B_{Emp}. Note that relation *Emp* is used as a domain for the "Tuple" attribute. This is only a convenient shorthand, not a nested attribute (cf. figure 1).

Relation name	*Emp*
Attribute name	*Domain name*
Id	*SURROGATE*
Name	*STRING(20)*
Mass	*KILOGRAM*
Height	*CM*
Salary	*DOLLAR*

Relation name	B_{Emp}
Attribute name	*Domain name*
Id	*SURROGATE*
Op	{*Ins, Del, Mod*}
Time	*TTIME*
Tuple	*Emp*

Figure 3: Schema for the user-defined relation, *Emp*, and its backlog, B_{Emp}.

□

The concept of backlog as presented in this subsection is inspired by the backlogs of the ADMS system [RK86], but while our backlogs are relations accessible through the query language, the backlogs of ADMS are purely implementation level concepts, and also ADMS does not support transaction time.

3.2 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:

$$\begin{aligned} R(t_{init}) &\stackrel{\text{def}}{=} R_{init} \\ R(t_x) &\stackrel{\text{def}}{=} R \text{ "at time } t_x\text{"}, \; t_x \geq t_{init} \\ R &\stackrel{\text{def}}{=} R(NOW) \end{aligned}$$

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. Note, that this feature helps provide transparency to the naive user. 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 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 correctly updated by the DBMS before subsequent retrievals.

EXAMPLE: Figure 4 shows the extension of *Emp* at two points in time. The extension of the corresponding backlog is shown in figure 5.

Emp(NOW - 20 days)				
Id	*Name*	*Mass*	*Height*	*Salary*
"surrogate"	Mark	85.0	177	90 000
"surrogate"	Brown	80.0	178	32 000
"surrogate"	Jensen	88.0	188	10 000

Emp				
Id	*Name*	*Mass*	*Height*	*Salary*
"surrogate"	Smith	74.5	170	30 000
"surrogate"	Brown	85.5	178	32 000
"surrogate"	Jensen	87.0	188	11 000

Figure 4: Time-slices of a relation, where NOW = 4:00 pm, May 1. 1989.

□

B_{Emp}							
Id	*Op*	*Time*	Id_{Emp}	*Name*	*Mass*	*Height*	*Salary*
"surrogate"	Mod	0420891238	"surrogate"	Brown	80.0	178	42 000
"surrogate"	Ins	0408891034	"surrogate"	Brown	80.0	178	32 000
"surrogate"	Ins	0402891456	"surrogate"	Mark	85.0	177	90 000
"surrogate"	Mod	0420891245	"surrogate"	Brown	85.5	178	32 000
"surrogate"	Ins	0331891131	"surrogate"	Jensen	88.0	188	10 000
"surrogate"	Del	0415891209	"surrogate"	Mark	85.0	177	90 000
"surrogate"	Ins	0419890902	"surrogate"	Smith	74.5	170	30 000
"surrogate"	Mod	0501891555	"surrogate"	Jensen	87.0	188	11 000

Figure 5: A sample backlog relation for the relation $Emp(NOW)$ where NOW = 4:00 pm, May 1. 1989.

A view is time dependent if at least one of the relations and views it is derived from is time dependent. Otherwise it it 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. We define:

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

$$B_R \stackrel{\text{def}}{=} B_R(NOW)$$

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

EXAMPLE: We can use B_{Emp} to retrieve all the employees that, during the last month, had varying properties. This is easily done with the following query:

$$\sigma_{NOW-30\ days \leq Time \leq NOW \wedge Op = Mod} B_{Emp}$$

The exact meaning of the query will be made clear in section 4. The result of the query is given in figure 6.

□

$\sigma_{NOW-30\ days \leq Time \leq NOW \wedge Op = Mod} B_{Emp}$							
Id	*Op*	*Time*	Id_{Emp}	*Name*	*Mass*	*Height*	*Salary*
"surrogate"	Mod	0420891238	"surrogate"	Brown	80.0	178	42 000
"surrogate"	Mod	0420891245	"surrogate"	Brown	85.5	178	32 000
"surrogate"	Mod	0501891555	"surrogate"	Jensen	87.0	188	11 000

Figure 6: A backlog query and its result. NOW = May 11. 1989.

3.3 Logical and Physical Aspects

So far, we have presented several kinds of relations. To get a better understanding of these, let us characterize them according to two dimensions. The first is related to the question of physical storage. Traditionally, base relations are stored while views are computed or virtual. The second is related to the question of logical derivability. Base relations are not derivable from other relations, while views are. See figure 7 where generally accepted characterizations of base relation and view are summarized [Dat86, Ull82, Cod79].

Traditional relation concepts	
Concept	*Description*
base relation	Actual data are stored in the database. The relation physically exists, in the sense that there exists records in storage that directly represent the relation. The *Emp* relation is an example. A base relation cannot be derived from other relations. A base relation definition is part of the *schema.*
view	A view is characterized as a *virtual, derived* or *computed* relation. It is not physically stored, but looks to the user retrieving information from the database as if it is. A view definition is part of a *subschema.*

Figure 7: The usual definitions of relation types.

In DM/T the relation concepts have new meaning. Every base relation has a backlog. All data of base relations are stored in the backlogs in the form of change requests. This makes backlogs act as base relations and base relations act as views, derived from backlogs. The new meanings are described in figure 8. In the sequel, we will use the definitions presented there.

As noted in figure 8 the system has the capability of storing views and base relations. When the system chooses to do so, it can be done in two ways. First, a view can be stored as a pointer

Redefined relation concepts	
Concept	*Description*
backlog	Backlogs are the relations that now function as base relations in the sense that they are stored and that all other relations (ultimately) are derived from these.
base relation	Base relations are not necessarily stored, and they are derived (directly) from backlogs. Thus the base relation *Emp* is derivable from B_{Emp} and is not necessarily physically existent.
view	The data of a view still is - directly or indirectly - construable from base relations - or backlogs. However, even though a view still is derived, it is not necessarily virtual or computed. Views can be persistent.

Figure 8: Redefinitions of relation types.

array, where an entry contain pointers to the relations (view or base) the view is derived from. The schema entry for a view contains information on how to materialize the view from the data pointed to (recursively). This structure is called an *index cache* (or just *cache* for short) because it inherits characteristics from both caches and indexes [Rou89, JMR89]. Second, a view can be stored as actual materialized data.

3.4 Summary of Data Structures

We can summarize the concepts presented in this section as illustrated in figure 9.

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

Figure 9: Different types of relations.

We distinguish between *backlog views*, traditional *views* and *base relations*. The only difference between views and base relations are that the former are derived indirectly from backlogs while the latter are derived directly. A view is valid only at a single point in time: The time-value specified when it was produced using the query language. Backlogs have an associated lifespan: From the time when the corresponding base relation was created till the current time - if they still exist - or otherwise till they were deleted. Backlog views inherit this notion of lifespan.

The second dimension in figure 9, *time dependence*, distinguishes between *fixed* and *time dependent* views. 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, however, be time dependent. A time dependent base relation can be visualized as a view that slides along a backlog as time passes. Similarly a backlog view can be thought of as a filtering window where one or both ends (start time and end time) move along a backlog. Let E be an expression which maps into the domain $TTIME$ and R a relation. For each time dependent time-slice, $R(E(NOW))$, there is a differential file, $\delta R(E(NOW))$. This differential file is a sequence of change requests in the backlog of the relation, that are not yet reflected in the actual state of the time-slice [RK86].

4 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 subsection 4.1 we present and shortly discuss the operators of our data model. In subsection 4.2 we illustrate the utility of parts of the query language.

4.1 Operators and Notation

In figure 10 we present the basic operators of the query language. Since base and schema relations must be time-sliced before they can be arguments of algebra operators, and since backlog relations are already "flat" in the same sense, the operators of the standard relational model work in our setting without modifications. This simplicity is a major advantage of our design. We will not discuss the operators of figure 10 in further detail.

From a conceptual point of view user-defined relations are historical, i.e. they have transaction time attributes. These attributes are crucial for time-slicing to be meaningful. The combination of the facts that we only manipulate time-sliced relations and that we want to comply with the *transparency principle* has resulted in the choice of hiding transaction time attributes in user-defined and schema relations. A transaction time attribute can be displayed by means of an explicit projection. In backlog relations time stamp attributes are displayed. A projection is required to remove such an attribute.

The first extension of the query language is the *unit* operator, Υ. It is used for changing units and precision of attribute domains. In this paper we focus on the precision aspect of the operator and we will not discuss the important aspects of conversion of compatible units. Instead we choose to refer

Standard Operators of the Query Language		
Notation	*Name*	*Description*
$R(t_x)$	Time-slice	This operator already was introduced and discussed earlier in the paper. In the literature the notation $\tau_{t=t_x}\ R$ is common. This notation is slightly more general, because time intervals can be easily expressed. In the present setting where only points in time are meaningful, the function application notation is the most convenient.
π	Projection	The standard projection operator. In the context of backlogs we will use π_{Tuple} to mean projection on all attributes of the associated user-defined or schema relation.
σ_F	Selection	The standard selection operator. The condition F can contain an arbitrary subquery.
$\times$	Cartesian product	The standard cartesian product operator.
-	Difference	The standard difference operator.
$\cup$	Union	The standard union operator.
$\cap$	Intersection	The usual definition applies: $R \cap S \stackrel{\text{def}}{=} R-(R-S) = S-(S-R)$
$\bowtie_F$	(Theta) Join	The standard definition: $R \bowtie_F S \stackrel{\text{def}}{=} \sigma_F, R \times S$. If no condition F is specified natural join is assumed.
$\rhd_F$	Semi join	$R \rhd_F S \stackrel{\text{def}}{=} \pi_{Att(R)}(R \bowtie_F S)$, where $Att(R)$ is the attributes of R.
"aggregate functions"		We allow for a full range of aggregate functions: max, min, mean, count, avg, sum, product, unit.

Figure 10: Notation, names and descriptions of basic operators.

to [KLI78] and [Geh82], from where 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:

$$\Upsilon_{A_{i_1}=D^x_{i_1},\, A_{i_2}=D^x_{i_2},\ldots,\, A_{i_k}=D^x_{i_k}}\ R(A_1 : D_1,\ A_2 : D_2,\ \ldots, A_n : D_n)$$

where $1 \leq i_j \leq n$, $j = 1, 2, \ldots, k$.

The result of this query is relation R with domain D_{i_1} of attribute A_{i_1} changed to $D^x_{i_1}$, domain D_{i_2} of attribute A_{i_2} changed to $D^x_{i_2}$, ..., domain D_{i_k} of attribute A_{i_k} changed to $D^x_{i_k}$.

A special relation, *Units*, contains information about which domains are compatible and how to

make the transformations. It has four attributes: $Domain_1$ and $Domain_2$ are each character strings listing domain names. *Factor* is a real number, and *Decimals* is an integer. See figure 11.

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

Figure 11: The relation *Units*.

The j'th entry in the *Unit* relation tells that an element in domain D_{i_j} can be transformed into an element in domain $D^x_{i_j}$ by multiplying it with k_{i_j} and, when representing the element in 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 in a coarser domain.

EXAMPLE: If we are measuring the masses of individuals in a population of people and want to use grams as the *unit of measure* we might define $D = \{500, 501, \ldots, 400000\}$. The domain defines the precision of measurement. If we change the *granularity* of data we get a new domain. For instance, the granularity kilo of data can be achieved by noting that 1 kilo = 1000 units of measure. Thus, $D' = \{1, 2, \ldots, 400\}$. The corresponding entry of the relation *Units* is (D, D', 1/1000, 0) □

In connection with the domain *TTIME* we have chosen the default unit 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*. These units all have zero decimals. If possible without causing ambiguity we do not distinguish between the domain *TTIME* and its compatible domains.

In our relations we record time stamps, i.e. simple time values. Sometimes it is convenient to have the corresponding time intervals. Thus we provide a *fold* operator (ϕ) and an inverse *unfold* operator (ϕ^{-1}) [LJ88]. The ϕ operator can be used on an unfolded relation, and it transforms the attribute defined over the domain *TTIME* into two attributes, *From* and *To*, both defined over the same domain. Used on an already folded relation it produces the identity. Similarly, ϕ^{-1} produces the identity when applied to an already unfolded relation. Figure 12 shows a sample query and its result; see also figure 5.

$\phi_{Time}\ \pi_{Name,\ Time,\ Mass,\ Salary}\ Emp(0410891600)$				
Name	*From*	*To*	*Mass*	*Salary*
Brown	0408891034	0420891238	80.0	32 000
Mark	0402891456	0415891209	85.0	90 000
Jensen	0331891131	-	88.0	10 000

Figure 12: Retrieving af folded relation. The query is issued during April 28.

The query in figure 12 first time-slices *Emp*; then attributes are projected - note that *Time* is projected; finally the attribute *Time* of the qualifying tuples is expanded into attributes *From* and *To*.

The next extension is the time related *when* operator, Ω [CUT85]. It is used for retrieval of times when a specified condition was true. If we issue the following query at May 1. 1989, 4:00 pm we get the result 0408891034, see figure 5:

$$\Omega_{Name=Brown \wedge Salary=32000}\, Emp$$

The attribute name of the unary relation returned is *From*[2].

If the Ω operator is used on a folded relation it returns intervals of validity. The following query issed at the same time as the one above returns the interval 033189 - 050189.

$$\Upsilon_{From=Day,\ To=Day}\ \ \Omega_{Name=Jensen \wedge Salary=10000}\ \ \phi\ Emp$$

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

The final operator is an *aggregate formation* operator, ξ. This operator is used to apply aggregate functions, see figure 10, to groups of attribute values. The operator we present here is a variation of the one described in [Klu82] and [Agr87]. The following notation is used:

$$\xi_{X,\ att\_name=agg\_fct}\ R$$

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 derived the following way: First, the tuples of R are divided into the groups implied by X. Second, *agg_fct* is applied to each group and the resultant value is associated with each tuple in the group as a value of the *att_name* attribute.

[2]Using standard operators this can be expressed as:

$$\pi_{Time}\ \sigma_{Name=Brown \wedge Salary=32000}\ \ B_{Emp}$$

If we assume that employees in the relation *Emp* belong to one of say three departments, the following query could generate a relation telling what the average salary is for each department:

$$\xi_{Dept,Avg\_sal=avg_{Salary}}\ Emp$$

where *Dept* is the assumed department attribute and *Avg_sal* is the new attribute to be generated. One group is generated for each distinct value of *Dept*. In general, X can be any sequence of distinct attributes in relation R. In addition, the keyword *Interval* can be substituted for Xmeaning that intervals previously generated by the Σ operator to be described in the next section are the groups to be used.

4.2 Sample Queries

We present a sequence of queries gradually getting more and more complicated. At 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. Finally, we show some queries using the new operators.

Simple retrieval involving base relations. The following queries demonstrate how time-sliced base relations are retrieved.

$$Emp(0501891600) \tag{1}$$

$$Emp(NOW) \text{ or, alternatively: } Emp \tag{2}$$

$$Emp(NOW - 20\ days) \tag{3}$$

Query (1) retrieves all employees as of close of business May 1. 1989 and is an example of a fixed time-slice. Figure 4 shows the result of the query. The queries (2) both retrieve the current state, i.e. the current employees as of the time of the queries. The second provide transparency. Finally, the last query (3) illustrates a more complicated time-dependent retrieval.

Retrievals involving base relations and views.

$$Rich\_Emp(t) = \sigma_{Salary \geq 40000}\ Emp(t) \tag{4}$$

$$\sigma_{Height \geq 180}\ Rich\_Emp(050189) \tag{5}$$

$$\sigma_{Salary \geq 80000}\ Rich\_Emp(NOW) \tag{6}$$

In (4) 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 database system. 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 (5) all tall and rich employees in the company as of May 1. 1989 are retrieved. This is a fixed time-slice that involves several levels of computation. First, *Rich_Emp* is substituted for its definition. Then, second, the time-slice is computed. Third, the selections are performed. In the last query (6) all currently very rich employees are retrieved. In contrast to the previous query the extension of this query is not stable.

Retrievals involving backlogs. Before we turn our attention toward queries directly involving backlog relations let us look at how the usefulness of backlogs supplement that of usual relations.

Suppose we want the changes to *Emp* between t_x and t_y. The following are all plausible candidate queries:

$$Emp(t_y) - Emp(t_x) \tag{7}$$

$$\sigma_{t_x \leq Time \leq t_y} B_{Emp} \tag{8}$$

$$\pi_{Tuple} \sigma_{t_x \leq Time \leq t_y} B_{Emp} \tag{9}$$

$$\pi_{Tuple} \sigma_{t_x \leq Time \leq t_y \wedge (Op=Ins \vee Op=Mod)} B_{Emp} \tag{10}$$

The result of the first query (7) is the tuples in *Emp* at t_y not in *Emp* at t_x. This does not tell us what took place between t_x and t_y. We will not retrieve any deletions that might have taken place in the time interval, for example. If we really want to know what took place between t_x and t_y we

would be better off using the backlog of *Emp*. We have to make clear what we precisely want. Let us look at some possibilities. The query (8) retrieves all possible information about what happened to *Emp*. Insertions, deletions and modifications are distinguished and the times when the requests were placed are available. Query (9) eliminates the special backlog attributes from the result. Thus several changes back and forth between identical *Emp* tuples will be eliminated and it will not be possible to distinguish between operation types any more. Finally, in (10) we have retrieved all employees that "changed", either because they were hired or because their previous properties were updated.

The possibilities listed are by no means exhaustive but illustrate sufficiently how a large number of detailed queries are easily formulated using backlogs.

Retrievals involving backlogs, views, and new operators. Let us now take a look at other kinds of queries.

$$\min_{Time}\ \sigma_{043089 \leq Time \wedge Op=Del}\ \Upsilon_{Time=Day}\ B_{Emp} \tag{11}$$

$$Emp(t_x) \rhd \pi_{Tuple}\ \sigma_{t_a \leq Time \leq t_b \wedge Op=Mod}\ B_{Emp} \tag{12}$$

$$Emp(t_x) - \pi_{Tuple}\ \sigma_{t_a \leq Time \leq t_b \wedge Op=Mod}\ B_{Emp} \tag{13}$$

$$\pi_{Name,\ Time}\ \Upsilon_{Time=Hour}\ \sigma_{27128800 \leq Time \leq 28128800 \wedge Op=Mod}\ B_{Emp} \tag{14}$$

In query (11) we use the $\min_{Time}$ operator to find the first employee to leave the company after April 30, 1989. Since change requests are assumed to be ordered according to time stamps the system need not first retrieve all *"Del"* change requests inserted after April 30. and then find the one with the smallest time stamp value. Instead the qualifying tuple can be retrieved directly, and a potential set valued retrieval is avoided. To find all the employees at t_x that changed properties between t_a and t_b we issue query (12) which involves both a time-sliced base relation and a backlog. The compliment is given by query (13). The query (14) results in a list of (*Name*, *Time*) of employees with property changes on December 27, 1988, but with a time granularity of one hour. It is assumed that the tuple (*TTIME*, *Hour*, 1/60, 0) is present in *Units*. We achieve a coarser granularity. Thus, if an employee changed salary more than once within the same hour (e.g. as a result of a typing error and a following correction) this will not be visible in the result.

Retrieval of patterns and exceptions. As the last example of this section let us illustrate how patterns and exceptions can be retrieved. In the next section we will present the operator Σ

which can be used for retrieval of patterns and exceptions as well. As will then be seen new classes of patterns and exceptions can be easily retrieved.

Suppose we want the employees that changed salary about the average number of times during the last two years.

$$Q_1 = \pi_{Name,Count}\ \xi_{Name,Count=count_{Time}}\ \sigma_{Op=Mod \wedge NOW-2\ years \leq Time \leq NOW}\ B_{Emp} \tag{15}$$

$$Q_2 = avg_{Count}\ Q_1 \tag{16}$$

$$Q_3 = \pi_{Name}\ \sigma_{.8*Q_2 \leq Count \leq 1.2*Q_2} Q_1 \tag{17}$$

$$\pi_{Name}\ \sigma_{Op=Mod \wedge NOW-2\ years \leq Time \leq NOW}\ B_{Emp}\ -\ Q_3 \tag{18}$$

$$\pi_{Name} \sigma_{0 \leq Count \leq .2*Q_2} Q_1 \tag{19}$$

In (15) we count, for each employee, the number of times the salary was changed within the given 2 year period. The attributes of the view are named *Name* and *Count*. This is an example of a sliding time window on a backlog screening out and aggregating relevant tuples. In (16) we find the average number of changes. In (17) we finally compare the number of changes of an employees salary with the average number of changes and keep employees that are close. We retrieve the employees with abnormal change pattern by (18). If we only want the employees with very few salary changes during the last 2 years, we can issue query (19), using Q_1 and Q_2.

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

5 Query Language Extension based on Compactness and Uniformity

This section presents the Σ operator. 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 and uniformity 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.

5.1 Motivation

A major, increasing problem in database systems is that they contain more information than the users have time to consume. New detailed information is entered faster than it can be looked at. This is the case in many applications, e.g. satelite surveillance, database monitoring, etc. Thus, the need to extract relevant and representative information from the database becomes still more important. 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.
- 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 TTIME of DM/T backlogs, it will be possible to answer queries like these:
  - When has there been 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.

5.2 Compactness and Uniformity of Domains

Let a relation S be given as follows

$$S(C_1, C_2, \ldots, C_l, D_1, D_2, \ldots, D_n)$$

We assume that domain names are unique and omit attribute names. We are only interested in domains for which there exist a metric and a total order as defined below[3]. Let the "D" domains be such ones and let

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

[3] Compare this distinction to the distinction in statistical and scientific databases between category data (measured data) and summary data (parameter data) [SW85]. The domains of interest include all summary data and some category data.

We drop the top and/or bottom indices in the sequel if possible without causing ambiguity. Associated with each domain, D_i, 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 of D_i, $i = 1, \ldots, n$, $dist_i$ and $\geq_i$, respectively.

DEFINITION: Let X be a set. Then a real valued function on $X \times X$, *dist*, is a **metric** iff it satisfies

$$dist(\alpha, \alpha) = 0$$

$$dist(\alpha, \beta) > 0 \text{ if } \alpha \neq \beta$$

$$dist(\alpha, \beta) + dist(\beta, \gamma) \geq dist(\alpha, \gamma)$$

$$dist(\alpha, \beta) = dist(\beta, \alpha)$$

THEOREM 5.1 *For any aggregated domain $D_{i_1} \times \cdots \times D_{i_k}$ the induced metric, dist, given below is indeed a metric.*

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

PROOF: Obvious. □

THEOREM 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.*

PROOF: Obvious. □

DEFINITION: $\geq$ is a **total order** on a domain D iff for all elements in D

$$\alpha \geq \alpha$$

$$(\alpha \geq \beta) \wedge (\beta \geq \gamma) \Rightarrow \alpha \geq \gamma$$

$$(\alpha \neq \beta) \wedge (\alpha \geq \beta) \Rightarrow \beta \not\geq \alpha$$

$$\forall \alpha, \beta \in D : (\alpha \geq \beta) \vee (\beta \geq \alpha)$$

In the following we feel free to use the derived predicates $\leq$, $<$, and $>$ if convenient.

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

$$\alpha = (\alpha^{i_1}, \alpha^{i_2}, \ldots, \alpha^{i_k})$$

$$\beta = (\beta^{i_1}, \beta^{i_2}, \ldots, \beta^{i_k})$$

$$\alpha \geq \beta \stackrel{\text{def}}{\Leftrightarrow}$$

$$\alpha^{i_1} >_{i_1} \beta^{i_1} \vee (\vee_{l=2}^{k}(\wedge_{j=1}^{l-1}(\alpha^{i_j} =_{i_j} \beta^{i_j}) \wedge \alpha^{i_l} >_{i_l} \beta^{i_l})) \vee \wedge_{j=1}^{k}(\alpha^{i_j} =_{i_j} \beta^{i_j})$$

The lexicographic order is a total order.

PROOF: It is easily seen that $\geq$ is indeed total. □

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

DEFINITION: The active domain, $D(t)$, of domain D in a relation S is **compact** iff

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

where $\geq$ is the order on D.

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

REMARK: Compactness is not preserved under subsequence formation. E.g. for some t, let $D(t) = D_1(t) \times D_2(t) \times \ldots \times D_n(t)$ be compact. Then, for the same t, $D'(t) = D_{i_1}(t) \times D_{i_2}(t) \times \ldots \times D_{i_k}(t), 1 \leq i_1 < i_2 < \cdots < i_k \leq n$ is not generally compact.

Let us illustrate this point by means of an example. We define

$$D = \{1, 2, 3, 4, 5\} \times \{1, 2, 3, 4, 5\} \times \{1, 2, 3, 4, 5\}$$

$$D(t) = \{ (1,2,4), (1,2,5), (1,3,1), (1,3,2) \}$$

It can be seen that $D(t)$ is compact. Now we project out the second domain to get $D'(t)$

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

This domain is *not* compact, because $(1,3)$ is missing.

A **maximal compact domain** is a compact domain to which no other domains can be aggregated without the resultant domain loosing the property of being compact. A relation can have many maximal compact domains, some of which might involve fewer domains, D_i, than do other non-maximal compact domains.

So far, the set of values of a domain present in a relation, i.e. the active domain, must constitute one single interval in order to be compact. We now generalize by relaxing this to a set of intervals and imposing various restrictions on this set instead. Let us define the concept of *partially compact domain.*

DEFINITION: The set of values of a domain D in a relation S, $D(t)$, is a **partially compact domain** iff all values of $D(t)$ together constitute a set of intervals. The elements of each interval constitute a compact domain. It is possible to impose various restrictions on the set of intervals:

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. 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.

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

Partial compactness is a generalization of compactness because 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 aggregated domains is not in general preserved under permutation of the aggregation sequence.

EXAMPLE: Let us, by means of an example, investigate why the definition as it is given above does not have this property. Let an aggregated domain be given by

$$D = D_1 \times D_2$$

and let the aggregated "interval" be made from the values

$$x_1, \ldots, x_n \text{ and } y_1, \ldots y_n.$$

Let us assume that (x_{i_1}, y_{i_2}) and (x_{j_1}, y_{j_2}) are arbitrary elements of an interval of $D(t)$. Assume without loss of generality that the former element is smallest. Then these elements belong to the interval:

$$(x_{i_1}, \{y_{i_2}, \ldots, y_m\}), (x_{i_1+1}, \{y_1, \ldots, y_m\}), \ldots, (x_{j_1}, \{y_1, \ldots, y_{j_2}\})$$

The reader may convince himself that the existence of these elements in an interval of $D_1(t) \times D_2(t)$ are *not* enough to make the corresponding elements of $D_2(t) \times D_1(t)$ an interval.

To be even more concrete let $D_1 = \{1, 2, 3\}$ and let $D_2 = \{a, b, c\}$ 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. Finally, observe that the permutation of the compact domain containing $(1, c)$ and $(2, c)$ as end points is not compact. This tells that not even the further restriction that the end points be mutually ordered according to the orders of their respective domains ensures compactness of a permuted domain. □

As in the case of simple compactness we talk about **maximal partially compact domains**. A (constrained) partially compact domain is maximal iff it cannot be expanded (under the given constraints). A relation can have a lot of both partially compact domains and maximal partially compact domains.

The definitions of compactness rely heavily on the notion of order. Every atomic (non aggregated) domain has an order. The lexicographic order on an aggregated domain induced by the orders of its constituent atomic domains is a generalization of the order of numbers (from digits) and words (from letters). It assigns a monotonic decreasing importance to atomic values of a domain. The first elements are the most significant, the second ones are the next most significant, and so on. For example, "el" comes before "le" and "bz" is less than "cc".

We now arrive at the concepts of *uniform domain* and *partially uniform domain.*

DEFINITION: $(D_1(t), D_2(t), \ldots, D_n(t))$ is a **uniform compact domain** iff it is compact and for each subdomain, $D_i(t)$, the number of times each value of $D_i(t)$ occurs is the same.

Similarly, $(D_1(t), D_2(t), \ldots, D_n(t))$ is a **uniform, partially compact domain** iff it is partially compact and for each subdomain, $D_i(t)$, the number of times each value (whatever interval it might belong to) of $D_i(t)$ occurs is the same.

A partially compact domain is a **partially uniform, partially compact domain** iff for each subdomain, $D_i(t)$, and for each interval in $D_i(t)$, the number of times each value occurs is the same.

Similarly to what we did for compact domains we also define maximality of the three kinds of uniform domains. A **maximal uniform compact domain** is a uniform maximal compact domain. A **maximal uniform, partially compact domain** is a uniform maximal partially compact domain. A **maximal partially uniform, partially compact domain** is a maximal partially compact domain that is partially uniform.

5.3 A Notation for Compactness Queries

We need a notation for expressing queries based on compactness. That is the subject of this subsection. We introduce a Σ operator that 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 that can be any combination of conjunctions, disjunctions, and negations of restrictions of the variables ι, δ and $\#$, as defined by the following grammar:

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

The E expression restricts the process of generating intervals: Intervals are only generated if they fulfill the specified restrictions.

EXAMPLE: The E expression below makes the Σ operator 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 the empty relation is returned. □

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

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

then any sequence A_{i_1}, A_{i_2}, ... , A_{i_k}, where $1 \leq i_1 < i_2 < \ldots < i_k \leq n$, 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 to be returned. The top-level syntax is

$$O \rightarrow X : Y$$

X is a specification of which 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.

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

EXAMPLE: The expression "max range: card" results in only the cardinality of the ones of the computed intervals with the largest range being returned. 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. □

Figure 13 defines the schemas of the various types of results that can be returned. If combinations of the three keywords are specified, the schemas are simply aggregated.

data			
$A_1 : D_1$	$A_2 : D_2$	...	$A_n : D_n$

range							
A_{i_1}.From : D_{i_1}	A_{i_2}.From : D_{i_2}	...	A_{i_k}.From : D_{i_k}	A_{i_1}.To : D_{i_1}	A_{i_2}.To : D_{i_2}	...	A_{i_k}.To : D_{i_k}

card
Card

Figure 13: Schema of the results when either *data*, *range*, or *card* is specified.

5.4 Sample Compactness Queries

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

The query (20) below retrieves all the tuples of B_{Emp} if the restrictions on ι, #, and δ are fulfilled.

$$\Sigma_{[E][Time][all:\ data]}\ B_{Emp} \tag{20}$$

The following algorithm will produce the result:

---

init # boundaries { initialization of restrictions implied by E }
init δ boundaries
init ι boundaries
init Res { the result relation is initially empty }
init i { the number of generated intervals, initially 1 }
init $\delta[i]$ { the size of the i'th interval, initially 0 }
init $\#[i]$ { the cardinality of the i'th 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_{Emp} 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) immediate successor of e
do e ← succ(e)
  $\#[i]++$
  Res ← Res ∪ e
check # boundaries
if unread elements in Time
then $i++$
check ι boundaries
compute $\delta[i]$
check δ boundaries

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

Now let us consider some more interesting examples.

$$\Sigma_{[\iota=1][Height][all:\ data]}\ Emp \tag{21}$$

$$\Sigma_{[\,][Time][min\ card:\ data]}\ \Upsilon_{Time=Hour}\ \sigma_{NOW-12\ months\leq Time}\ B_{Emp} \tag{22}$$

$$\Sigma_{[\,][Time][\#>X:\ range]}\ B_{Emp} \tag{23}$$

$$\Sigma_{[\,][Time][max\ range,\ max\ card:\ range]}\ \sigma_{Op=Ins}\ B_{Emp} \tag{24}$$

$$\Sigma_{[\iota\leq 10\wedge 8\leq\delta\wedge 1000\leq\#][Time][all:\ data]}\ \sigma_{NOW-2years\leq Time}\ B_{Emp} \tag{25}$$

$$\Sigma_{[4\geq\iota][Time][all:\ range,card]}\ \sigma_{Op=Del\wedge 09058916\leq Time\leq 10058907}\ \Upsilon_{Time=Hour}\ \sigma_{Op=Del}\ B_{Emp} \tag{26}$$

$$Avg_{Sum}\ \pi_{Sum}\ \xi_{Time,\ Sum=count_{Time}}\ \Upsilon_{Time=Day}\ \sigma_{Op=Day}\ B_{Emp} \tag{27}$$

Statistical normal forms based on compactness and uniformity are important in statistical applications of contents of databases [Gho89]. A relation is in 1. statistical normal form (1.SNF) if every atomic non-category domain is compact [Gho89]. Let us illustrate how we can test whether a relation is in 1.SNF. In *Emp* this means that the active domains of attributes Mass, Height, and Salary must be compact. Query (21) returns the whole relation *Emp* if Salary is compact and the

empty relation otherwise. Using similar queries for Mass and Height we can determine whether *Emp* is compact. Given a relation in 1.SNF a natural continuation would be to ask if it is also in 2.SNF, i.e. whether every atomic non-category domain is also uniform. This can be done using the aggregate formation operator and count functions. We have chosen not to present an operator based on the notion of uniformity.

The query (22) retrieves the data of the hour(s) during the past year where the least happened to relation *Emp*. First, the part of the backlog of *Emp* for the past year is selected. Second, the time unit is changed to *Hour*. Third, the tuples of B_{Emp} are grouped according to the Time attribute and the elements of the interval with the least number of elements is returned as the final result.

In query (23) intervals are again generated on the basis of the attribute Time of B_{Emp}. This time the ranges of the cardinality-wise large intervals (i.e. intervals with more than X elements) are the result.

In the next query (24) we find the periods through the lifetime of *Emp* where the most insertions took place. Among all possible intervals the ranges of only the ones with largest cardinality and range are returned.

In (25) we put some restrictions on the interval generation process. For simplicity, we have chosen those of the first example of subsection 5.3. If it is possible to generate intervals under the imposed restrictions the whole relation is returned; otherwise B_{Emp} with an empty extension is the result.

In query (26) we are interested in deletions to the database that took place after 4 pm on May 9. and before 7 am on May 10. Before the Σ operator is applied we select the interesting parts of the backlog and change the domain of attribute Time. Then, if at least four intervals will result, the ranges and cardinalities if the generated intervals are returned; otherwise the empty relation is the result.

The final query (27) returns the average number of deletion requests per hour to the relation *Emp*.

In conclusion, we have illustrated the utility of the Σ operator. The reader may compare the sample queries to the informal questions of subsection 5.1.

6 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 focussed on abstracting common patterns and exceptions in the change history.

First, we added transaction time to the standard relational model. We gave a precise characterization of the time concept chosen and argued that tuple time stamping better suited our objective to provide a not just small but also transparent extension than did attribute value time stamping.

Second, we extended the data structures of the standard relational model to include system generated and maintained backlog relations that proved to be ideal for recording a detailed change history. The backlogs contained requests for changes to their associated base relations rather than the results of the requests. Thus they more faithfully reflected the evolution of the database. Change requests were grouped into requests for insertions, deletions, and modifications. Using only the standard relational algebra the extended data structures allowed for historical queries.

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 Σ operator to the query language. Based on the notion of compactness, the operator provides a new way to group data. Usually data were grouped according to predefined groups. The idea of the Σ operator was to let the data themselves determine the groups. Thus the operator grouped consecutive data together, and started a new group when a hole was detected. It was possible to impose restrictions on the grouping process. The number of allowed groups could be specified; the allowed widths and cardinalities of the groups could be specified. Not only could the grouping process be applied to atomic attributes, but it could be applied to arbitrary sequences of domains, i.e. aggregated domains. Also, it was possible to select from among the generated groups the ones to be the result of an application of the operator. We demonstrated the use of the operator by sample queries. The *unit* operator allowed for varying the consecutiveness of data, and the *aggregate formation* operator allowed for computing statistics on generated groups. As a whole the query language used in conjunction with the extended data structures allowed for retrieving patterns and exceptions in the change history of the database.

The incorporation of a general and flexible mechanism for version support into the data model of this paper is subject of current research. So is further investigation of the query evaluation subsystem underlying the data model of this paper [JMR89].

While the basic idea of the operator Σ is fixed, an issue of further research is how to apply it in different settings, and a more extensive study of the applications of the basic idea can lead to other

versions of the operator, better suited to their 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 by allowing for interactive modification of the subscripts E and O is a natural future direction with some resemblance to query generalization [Mot84a, MDT, Mot88, Mot84b]. Also, the application of the operator to the statistical normal forms of [Gho89] is an interesting direction, and finally it would be of interest to investigate the possible application of a Σ-like operator to dependency theoretic problems.

Acknowledgements

We would like to thank Kathleen Romanik for carefully reading parts of the paper and supplying valuable comments.

References

[Agr87] Rakesh Agrawal. Alpha: an extension of relational algebra to express a class of recursive queries. In *Proceedings of the 3rd International Conference on Data Engineering, Los Angeles, California*, pages 1–11, February 1987.

[Ari86] Gad Ariav. A temporally oriented data model. *ACM Transactions on Database Systems*, 11(4):499–527, December 1986.

[BADW82] A. Bolour, T. L. Anderson, L. J. Dekeyser, and H. K. T. Wong. The role of time in information processing: a survey. *ACM Sigmod Record*, 12(3):27–50, April 1982.

[BMP] Ben-Ari, Manna, and Pnueli. *The Temporal Logics of Branching Time.* , .

[CC87] James Clifford and Albert Croker. The historical relational datamodel (hrdm) and algebra based on lifespans. In *Third International Conference on Data Engineering*, pages 528–537, February 1987.

[Cha88] Surajit Chaudhuri. Temporal relationships in databases. In *Proceedings of the 14th International Conference Very Large Data Bases*, pages 160–170, 1988.

[Cod79] E. F. Codd. Extending the database relational model to capture more meaning. *ACM Transactions on Database Systems*, 4(4):397–434, December 1979.

[CUT85] James Clifford and Abdullah Uz Tanzel. On an algebra for historical relational databases: two views. In *Proceedings of the ACM Sigmod '85*, pages 247–265, 1985.

[CW83] James Clifford and David S. Warren. Formal semantics for time in databases. *ACM Transactions on Database Systems*, 8(2):214–254, June 1983.

[Dat86] C. J. Date. *An Introduction to Database Systems. The Systems Programming Series*, Addison Wesley Publishing Company, fourth edition, 1986.

[Gad88] K. Shashi Gadia. A homogeneous relational model and query languages for temporal databases. *ACM Transactions on Database Systems*, 13(4):418–448, December 1988.

[Geh82] Narain Gehani. Databases and unit of measure. *IEEE Transactions on Software Engineering*, SE-8(6):605–611, November 1982.

[Gho86] Sakti P. Ghosh. Statistical relational tables for statistical database management. *IEEE Transactions on Software Engineering*, SE-12(12):1106–1116, December 1986.

[Gho89] Sakti P. Ghosh. *Statistical Relational Databases: Normal Forms.* Technical Report, IBM Research Division, Almaden Research Center, San Jose, California, 1989. Submitted to ACM Sigmod '89.

[Jac83] Michael A. Jackson. *System Development. Prentice-Hall International Series in Computer Science*, Prentice-Hall International, Inc., 1983.

[JMR89] Christian S. Jensen, Leo Mark, and Nick Roussopoulos. *Incremental Implementation Model for Relational Databases with Transaction Time.* technical report TR-2275, UMIACS-TR-8963, Department of Computer Science. University of Maryland, College Park, MD 20742, June 1989. Submitted for publication.

[KLI78] Michael Karr and David B. Loveman III. Incorporation of units into programming languages. *Communications of the ACM*, 21(5):385–391, May 1978.

[Klu82] A. Klug. Equivalence of relational algebra and relational calculus query languages having aggregate functions. *Journal of The ACM*, 29(3):699–717, July 1982.

[Lam] Leslie Lamport. *Sometime is sometimes not never.* , .

[LJ88] Nikos A. Lorentzos and Roger G. Johnson. Extending relational algebra to manipulate temporal data. *Information Systems*, 13(3):289–296, 1988.

[Mal86] Francesco M. Malvestuto. Statistical treatment of the information content of a database. *Information Systems*, 11(3):211–223, 1986.

[Mat81] Lars Mathiassen. *Systems Development and Systems Development Method.* Aarhus University, 1981. In Danish.

[MDT] Amihai Motro, Alessandro D'Atri, and Laura Tarantino. The design of kiview: an object-oriented browser. In *?*, pages 107–131, ?.

[Mot84a] Amihai Motro. Browsing in a loosely structured database. In *Proceedings of ACM Sigmod '84*, pages 197–207, 1984.

[Mot84b] Amihai Motro. Query generalization. In *Proceedings of the First International Workshop on Expert Database Systems*, pages 314–325, October 1984.

[Mot88] Amihai Motro. Vague: a user interface to relational databases that permit vague queries. *ACM Transactions on Office Information Systems*, 6(3):187–214, July 1988.

[MS88] Edwin McKenzie and Richard Snodgrass. Extending the relational algebra to support transaction time. In *Proceedings of the ACM Sigmod '88*, pages 467–477, 1988.

[OO85] Gultekin Ozsoyoglu and Zehra Meral Ozsoyoglu. Statistical database query languages. *IEEE Transactions on Software Engineering*, SE-11(10):1071–1081, October 1985.

[RK86] Nick Roussopoulos and Hyunchul Kang. Principles and techniques in the design of *adms±*. *Computer*, 19(12):19–25, December 1986.

[Rou82a] Nick Roussopoulos. The logical access path schema of a database. *IEEE Transactions on Software Engineering*, 8(6):563–573, November 1982.

[Rou82b] Nick Roussopoulos. View indexing in relational databases. *ACM Transactions on Database Systems*, 7(2):258–290, June 1982.

[Rou89] Nick Roussopoulos. *The Incremental Access Method of View Cache: Concept, Algorithms, and Cost Analysis.* technical report UMIACS-TR-89-15, CS-TR-2193, Department of Computer Science, University of Maryland, College Park, MD 20742, February 1989.

[SA85] Richard Snodgrass and Ilsoo Ahn. A taxonomy of time in databases. In *Proceedings of the ACM Sigmod '85*, pages 236–246, 1985.

[Sho82] Arie Shoshani. Statistical databases: characteristics, problems, and some solutions. In *Proceedings of the Eighth International Conference on Very Large Data Bases*, pages 208–222, September 1982.

[SK86] Arie Shoshani and Kyoji Kawagoe. Temporal data management. In *Proceedings of the Twelfth International Conference on Very Large Data Bases*, pages 79–88, August 1986.

[Sno87] Richard Snodgrass. The temporal query language tquel. *ACM Transactions on Database Systems*, 12(2):247–298, June 1987.

[SW85] Arie Shoshani and Harry K. T. Wong. Statistical and scientific database issues. *IEEE Transactions on Software Engineering*, SE-11(10):1040–1047, October 1985.

[TAO89] Abdullah U. Tansel, Erol M. Arkun, and Gultekin Ozsoyoglu. Time-by-example query language for historical databases. *IEEE Transactions on Software Engineering*, 15(4):464–478, April 1989.

[Ull82] Jeffrey D. Ullman. *Principles of Database Systems. Computer Software Engineering Series*, Computer Science Press, second edition, 1982.

[UT86] Abdullah UZ Tansel. Adding time dimension to relational model and extending relational algebra. *Information Systems*, 11(4):343–355, 1986.

[UT87] A. Uz Tansel. A statistical interface for historical relational databases. In *Third International Conference on Data Engineering*, pages 538–546, February 1987.