UMIACS-TR-90-105
CS-TR-2516

August 1990

A FRAMEWORK FOR VACUUMING TEMPORAL DATABASES*

Christian S. Jensen
Leo Mark

Department of Computer Science
University of Maryland
College Park, MD 20742

Abstract

In conventional databases, the amount of data typically reaches a certain level and is then relatively stable. In databases supporting transaction time, old data are retained, and the amount of data is ever growing. Even with continued advances in mass storage technology, vacuuming (i.e., deletion or off-line storage of data) will eventually be necessary. At the same time, the fundamental principle, that history cannot be changed, of transaction time databases must be obeyed.

This paper provides a framework for vacuuming subsystems for relational transaction time databases. Our main focus is to establish a foundation for correct and cooperative query processing through the modification of queries that cannot be processed due to vacuuming. In doing this, we provide language facilities for specifying vacuuming; we present three classifications of vacuuming specifications; and we define correctness criteria for vacuuming specifications. Based on the classifications, we provide a comprehensive set of rules for expressing modified queries. For some of the classes, modified queries can be expressed using relational algebra—for others, this is impossible, and an extended, tagged relational algebra is used instead.

The framework is a useful tool for designers of specific vacuuming subsystems. The framework is presented in the context of a previously developed relational model with transaction time support, DM/T.

General Terms: Design, Algorithms

Additional Keywords and Phrases: relational model, temporal databases, transaction time, query modification, temporal data, irrelevant data

---

*This work is supported by NSF Grant IRI-8719458 and AFOSR Grant 89-0303. Jensen is, in addition, supported by Aalborg University, Denmark. Mark is also affiliated with the University of Maryland Institute for Advanced Computer Studies and Systems Research Center.
Correspondence via e-mail to jensen@brillig.cs.umd.edu.

1 Introduction

A conventional database management system (DBMS) differs significantly from a DBMS supporting transaction time. Relations of a conventional DBMS store a single snapshot, the current state. In contrast, relations of a DBMS supporting transaction time, termed roll-back relations, store all previous states using time stamps. Thus, it is possible to retrieve not only the current state, but any previous state. It is also possible to perform, for example, trend and exception analysis on the change history of relations. This dramatically increased functionality does, however, not come for free. With no deletions, roll-back relations are ever-growing and will eventually outgrow any mass storage device. In addition, efficiency degrades as relations grow. It is apparent that effective means of coping with growth is required before a temporal system can be put into practical and industrial use—this is the task of a vacuuming subsystem.

The most immediate motivation for a vacuuming subsystem is that storage is limited. It must be possible to remove data when they are no longer needed, or when additional, free space is needed for more important data. On the other hand, a vacuuming subsystem should also guard against loss of crucial data—simple cyclic removal schemes, deleting the oldest data when space for new data is required, are not sufficient. Another motive for vacuuming is the general tendency that the more data to manage, the less efficiently can it be managed. Limitation of storage and degrading performance are both *laws of nature*. *Laws of society* also motivate vacuuming. Existing laws require capabilities to delete certain records of previous history while preserving other records for periods of time. Similarly, capabilities are required to enforce *laws of business*, business policies.

A fundamental principle of a roll-back database is that the recorded history must not be changed [25]. Thus, the answer to a fixed query (e.g., "how many employees were recorded in the database as of January 22nd 1990?") must never change. Deletion of data by a vacuuming subsystem must obey this principle. It is done by modifying queries that would otherwise have accessed vacuumed data into corresponding modified queries that avoid data that have been vacuumed.

Previously vacuumed data can be involved in queries in several ways. For example, the selection criteria of a query, Q, can involve an attribute, the data of which have been removed, and some of the data requested by Q may be missing. A vacuuming subsystem must handle such situations cooperatively. Our approach is to *modify* the user-issued query, Q, into a set of modified queries, $M_V Q_i$, $1 \leq i \leq k$, where each query reflects that vacuuming, according to a vacuuming specification, V, has been performed and thus does not reference data that have been vacuumed. If Q is not equivalent to any of $M_V Q_i$, then Q can not be processed (due to vacuuming), and the best the system can do is to process one of $M_V Q_i$, which can be a generalized or specialized (or both) version of Q. In this case, the user is confronted with the set of alternative queries and asked to either select a query, upon which that query will be processed, or to issue another query.

To be appropriate, vacuuming specifications must be *growing* (i.e., data once specified for deletion must continue to be specified for deletion). Not all vacuuming specifications are appropriate. For example, a specification, stating that data can be deleted if they were inserted into the database between two and three years ago, does not make sense. Data more than three years old cannot be removed, and data between two and three years old eventually

become more than three years old. As a result, no data can be deleted. We term this kind of useless specification *moving*, and they are detected at specification time. In addition, a specification is inappropriate if it removes data needed to perform vacuuming according to other existing specifications; if it removes data needed to compute queries of application programs and views; and if it conflicts with integrity constraints. Inappropriate specifications are detected at specification time.

Vacuuming is performed by a vacuuming demon according to the specifications. While vacuuming logically has eager semantics, any degree of eagerness or laziness can be adopted for the actual physical removal of base data, and a variety of conditions triggering the demon can be employed.

In this paper, we provide a framework for vacuuming subsystems, and we describe how a vacuuming subsystem augments the functionality of a DBMS. In particular, we have chosen to concentrate on the fundamentals for providing correct and cooperative query processing of queries that reference data, missing due to vacuuming: we investigate the fundamental problem of how to express modified queries. In doing so, we identify 8 classes of vacuuming specifications. These classes are characterized according to two additional classifications that elicit important aspects of the nature of each class. Following this, each of the 8 classes are treated in turn. We illustrate the functionality of each class. We consider how to express, in a single expression, the vacuuming of a relation. This is the basis for modifying queries that access vacuumed relations. Also, we exemplify how to express vacuumed relations, and we exemplify querying of vacuumed relations.

We discuss vacuuming in the context of the data model DM/T [14, 13, 15, 16], but it should be clear that any temporal data model supporting transaction time can be augmented with a vacuuming subsystem, and that vacuuming facilities are necessary before such a model can be put into commercial use.

To our knowledge, the vacuuming subsystem (VS) framework presented in this paper represents the first work on systems of this nature.

The database management system, Postgres [22], has two features related to vacuuming. First, *cutoff points* for relations can be specified [28]. Consider two examples: `discard EMP before ''1 week''`, `discard EMP before ''now''`, and `discard EMP`. The former deletes data more than 1 week old from the relation, *Emp*; the latter two both retain only the current state. In addition, when a base relation is created, it can be tagged with the designation `no archive` (as opposed to `light archive` and `heavy archive`), meaning the same as `discard <rel-name>` [27]. Second, an asynchronous demon, the vacuum cleaner, is employed. A Postgres database is stored on both magnetical and optical disk, and the *vacuum cleaner* moves records of committed transactions from magnetic disk onto optical disk. By default, it moves records of all transactions committed at the time it is activated onto optical disk, but records can be kept on magnetical disk by delaying vacuuming. In [28], the syntax is `vacuum <rel-name> where <QUAL>` where `rel-name` is a base relation and `QUAL` a qualification, typically involving NOW. For example, `vacuum EMP where EMP.tmax` $\leq$ `NOW - 30 days` tells to retain EMP records on magnetical disk until they are 30 days old. In [27], the syntax is restricted to `vacuum <rel-name> after <T>`, and the example becomes `vacuum EMP after ''30 days''`. Vacuuming in the context of crashes is shown to be without complications, and the cost of vacuuming

is shown to be marginal.

Query languages of the standard relational model allow for set-at-a-time deletion. For example, in SQUARE [5]—a predecessor of SQL—the syntax is $\uparrow R_{A_1,\ldots,A_n}(< \mathbf{exp} >)$ where $A_1, \ldots, A_n$ are attributes of R and $< \mathbf{exp} >$ evaluates to some n-ary relation [29]. The result of executing the command is to delete from R all tuples where the values of attributes $A_1, \ldots, A_n$ match the corresponding attributes of the computed n-ary relation. This functionality, a special case of a VS's functionality, applies only to a standard relational model. Deletions in a model extended with transaction time needs extra attention because they must adhere to the principle of never changing the recorded history.

In *database restructuring* and *schema evolution*, identification of parts of data is used not only to remove data but, more typically to split and restructure relations. Schema evolution in a model extended with time has been addressed by [18]. In a standard relational model, vertical and horizontal partitioning of relations and dropping of attributes has been addressed by [23]. Our framework allows for pinpointing both vertical and horizontal fragments of relations, but only for the purpose of deleting attribute values—vacuuming does not result in restructuring of databases and evolution of schemas.

Query modification is a technique for modifying queries. This technique is used for implementing integrity constraints and views [26]. For example, an occurrence of a view name in a query is substituted by the definition of the view so that the resulting query only references the base relation(s) that are used to define the view. Modification of queries using equivalence transformations is also done for performance reasons during query optimization (e.g., [24, 30]). Modification of queries by a VS serves a different purpose: queries are modified into similar (as defined later), but not equivalent queries, in order to provide cooperative processing when queries rely on vacuumed data and therefore cannot be processed correctly as they are.

As an extension of query modification, techniques for *query generalization* can be used to increase the size of answers to queries [19, 6]. Similarly, *query specialization* can be used to decrease the size. In this paper, we consider only the foundation for correct and cooperative query processing. Based on this foundation, both techniques can be applied to further simplify modified queries and increase cooperativeness.

Vacuuming specifications somewhat resemble integrity constraints, but there are important differences as well. To illustrate some of these, consider *referential integrity* where the idea is that "if some tuple references another tuple, then this other tuple must exist." If, in the well-known suppliers and parts database [10], there is a shipment for supplier "s_1" in the shipment relation, then the supplier "s_1" must exist in the suppliers relation. To enforce referential integrity, an integrity subsystem must monitor update operations and take action if the updates violate integrity. Monitoring is expensive as it requires search of entire relations for matching foreign and primary keys—this is a popular explanation why many existing DBMS's do not support referential integrity [9]. A vacuuming subsystem does not have this efficiency problem as it needs not do any processing when update operations take place. Actual physical vacuuming needs not to be done eagerly; it can be done when convenient (i.e., vacuuming can be triggered by a low system workload, or by another condition). In addition, vacuuming is cheap in itself: no extra data need to be referenced at query processing time; only modification and perhaps reorganization of a query expression may

have to be done as extra work. Finally, at specification time, validity checks of specifications need to be performed. The checks involve only the database intension, no extension is involved.

When vacuuming occurrences of attribute values from tuples, ***missing values*** occur, and vacuuming-null values ("-") are used. Such values can be treated by DBMS's as they already treat traditional null-values—both signify missing information. Our research on vacuuming is separate from issues of null-value research.

Vacuuming concerns base data, and we have made the simplifying assumption that vacuuming is specified only in terms of backlog relations. This is done to avoid the issue of update through views [11, 12].

In the next section, we introduce a sample database and demonstrate the concept of vacuuming. In section 3, we define the extensions of functionality of a DBMS with vacuuming. In section 4, we present a taxonomy for vacuuming specifications. Then, in section 5, we define concepts necessary for the presentation in the sections to follow. Based on the taxonomy and the concepts, sections 6 - 10 discuss various kinds of vacuuming. The final two sections (11 and 12) contain the conclusion and topics of future research.

2 An Example

We devote this section to an example in order to convey the idea of a roll-back DBMS extended with vacuuming facilities. We use DM/T, a roll-back model with tuple stamping [14, 13], for this purpose. A brief introduction to DM/T is offered in the appendix.

Assume that we have a database, $\mathcal{DB}$, with a user-defined relation, *Emp*. The schema of *Emp*, $\mathcal{S}(Emp)$, is given as[1]

$$Emp(Id, Sal, Bal : INT; Sex : \{M, F\})$$

where *Sal* and *Bal* is the salary and account balance, respectively, of an employee with the identification number *Id*. Updates to *Emp* result in change requests being entered into the *backlog* of *Emp*, B_{Emp}. The backlog has the schema

$$B_{Emp}(Bid : SURROGATE; Op : \{I, D, M\}; Time : TTIME; Id, Sal, Bal : INT; Sex : \{M, F\})$$

where *Bid* is an identifier of backlog tuples; *Op* identifies tuples as either insertion (*I*), deletion (*D*), or modification (*M*) requests; and *Time* is a transaction time stamp. Note that B_{Emp} is the base relation, and because B_{Emp} records the complete change history of *Emp*, any past state of *Emp* can be derived from it (by means of a time slice).

All updates of *Emp* result in tuples being inserted into B_{Emp}, and nothing is ever deleted. Therefore, B_{Emp} is ever-growing, and it is likely that B_{Emp} will eventually contain some data that are irrelevant to the users. Now, assume that the current business policy is that data four years and older are not to be retained, that tuples between two and four years old with value *F* of the attribute *Sex* can be disregarded, but that a full record of employees

[1] $\mathcal{S}(R)$ denotes the schema of relation R. Similarly, $\mathcal{I}(R)$ denotes the set of instances of $\mathcal{S}(R)$. Informally, we employ R to mean an arbitrary, given instance of $\mathcal{S}(R)$.

deleted from the database must be retained. This can be specified by entering the following three lines:

$$V_1 \quad \rho(B_{Emp}) : \sigma_{Time \leq NOW-4yrs} B_{Emp}$$

$$V_2 \quad \rho(B_{Emp}) : \sigma_{NOW-4yrs<Time \leq NOW-2yrs \wedge Sex=F} B_{Emp}$$

$$V_3 \quad \kappa(B_{Emp}) : \sigma_{Op=D} B_{Emp}$$

Let the set $V = \{V_1, V_2, V_3\}$ constitute the current specification for B_{Emp}. It is read as follows: "remove (ρ) from B_{Emp} all tuples where the value of the attribute *Time* is less than the current time (NOW) minus four years; remove from B_{Emp} all tuples with *Sex* F where the value of *Time* is less than four and at least two years old; keep (κ) in B_{Emp} all tuples where attribute Op has value D". While V_1 and V_2 are removal specifications and tell what must be removed, V_3 is a keep specification and tells what must be kept.

When a specification for a backlog is created or updated, a VS will perform a list of checks to make sure that the specified removals make sense and fit harmoniously with other specifications, view definitions, the needs of application programs, etc. If conflicts are detected, the Data Base Administrator (DBA) is warned and may have to take action (section 3).

Above, we have considered the specification of vacuuming; below, we consider queries on relations that have been subjected to vacuuming. Assume that query $Q = \sigma_{Sal \geq 30k} B_{Emp}$ is issued against $\mathcal{DB}$ and denote the result $Q(\mathcal{DB})$. Then, assume that the specification V is in effect and evaluate Q, resulting in $Q(M_V\mathcal{DB})$. Observe that because Q can reference data vacuumed due to V, it is possible to have $Q(\mathcal{DB}) \neq Q(M_V\mathcal{DB})$. For example, a tuple of B_{Emp} more than four years old and with a *Sal* value of 40k will appear in $Q(\mathcal{DB})$, but not in $Q(M_V\mathcal{DB})$. This would be a violation of the principle that the history recorded in the database cannot be changed. Thus, query Q cannot be answered as it is. The system does not merely refuse to evaluate Q: it cooperates by modifying Q into a set of one or more queries, $M_V Q_i$, $1 \leq i \leq k$, that are equivalent[2], $\forall\, 1 \leq i, j \leq j\ (M_V Q_i \equiv M_V Q_j)$. These represent the result of Q when taking vacuuming according to V into consideration.

In our example, where Q is issued with specification V in effect, the best the system can do is augment the selection criterion of Q with the additional restrictions imposed by V and present the modified queries to the user as alternatives. It is easily seen that $M_V Q$ is of the form $\sigma_{F'} B_{Emp}$. If we let F, F_1, F_2, and F_3 denote the selection criteria of Q, V_1, V_2, and V_3, respectively, then

$$\begin{aligned}
F' &\equiv F \wedge [(\neg F_1 \wedge \neg F_2) \vee F_3] \\
&\equiv Sal \geq 30k \wedge [(\neg(Time \leq NOW - 4yrs) \wedge \neg(NOW - 4yrs < Time \leq NOW - 2yrs \wedge Sex = F)) \vee Op = D] \\
&\equiv Sal \geq 30k \wedge [(Time > NOW - 4yrs \wedge [(Time > NOW - 2yrs \vee Time \leq NOW - 4yrs) \vee Sex = M]) \vee Op = D] \\
&\equiv Sal \geq 30k \wedge [(Time > NOW - 4yrs \wedge (Time > NOW - 2yrs \vee Sex = M)) \vee Op = D]
\end{aligned}$$

[2]Let the schema, $\mathcal{S}(\mathcal{DB})$, and a vacuuming specification, V, of a database, $\mathcal{DB}$, be given. Two query expressions, Q_1 and Q_2, are *equivalent*, $Q_1 \equiv Q_2$, iff $\forall I \in \mathcal{I}(\mathcal{DB})\ (Q_1(I) = Q_2(I))$. In addition, Q_1 and Q_2, are *equivalent with respect to (w.r.t.) specification* V, $Q_1 \equiv_V Q_2$, *iff* $\forall I \in \mathcal{I}(\mathcal{DB})\ (Q_1(I_V) = Q_2(I_V))$.

Similarly, we can derive

$$\begin{aligned} F' \equiv\ & (Time \leq NOW - 4yrs \wedge Op = D \wedge Sal \geq 30k) \vee \\ & (NOW - 4yrs < Time \leq NOW - 2yrs \wedge (Sex = M \vee Op = D) \wedge Sal \geq 30k) \vee \\ & (NOW - 2yrs < Time \wedge Sal \geq 30k) \end{aligned}$$

When confronted with modified queries, the user can either select one of these for processing or change and reissue one of these (figure 1). It is guaranteed that a modified query returned by the system can be processed, but if a query is changed and reissued, the system may have to go through a new cycle of modification and display.

```
>> select[Sal >= 30k] B[Emp]
   query cannot be evaluated due to missing data; alternative, equivalent queries:
   select[Sal >= 30k and ((Time > NOW - 4yrs and (Time > NOW - 2yrs or Sex = M)) or Op = D)] B[Emp]
   select [(Time <= NOW - 4yrs and Op = D and Sal >= 30k) or
      (NOW - 4yrs < Time <= NOW - 2yrs and (Sex = M or Op = D) and Sal >= 30k) or
      (NOW - 2yrs < Time and Sal >= 30k)] B[Emp]
```

Figure 1: The result of issuing query $Q = \sigma_{Sal>30k} B_{Emp}$ with specification $V = \{V_1, V_2, V_3\}$ in effect.

The system divides the process of modifying a query into two parts. At specification time, a backlog expression (e.g., B_{Emp}) is modified by its current vacuuming specification, V, into a modified backlog expression, $M_V B_{Emp}$. For our example, we have

$$M_V B_{Emp} \equiv \sigma_{(Time>NOW-4yrs \wedge (Time>NOW-2yrs \vee Sex=M)) \vee Op=D} B_{Emp}$$

An expression like this describes a physically existing base relation—it tells exactly which data are present in the vacuumed B_{Emp}. Even though it can look rather complex, it is actually an atomic unit in the sense that optimization transformations are pointless when computing the extension. Then, at query time, the system needs only to replace occurrences of backlog expressions by corresponding modified expressions and possibly use equivalence transformations to expand the resulting expression into a set of equivalent expressions. In the query $\sigma_{Sal \geq 30k} B_{Emp}$, the occurrence of B_{Emp} is merely replaced by the expression for $M_V B_{Emp}$.

3 The Extended Architecture

This section describes the architecture of a DBMS extended with a VS and gives an overview of the functionality of such a system. More specifically, we first consider how a VS augments the query processing subsystem of a DBMS, and then we consider the specification of vacuuming itself.

In an extended system, all queries pass through a *vacuuming filter*. Queries that *do not* reference vacuumed data are passed on to the DBMS for processing; queries that *may* reference vacuumed data are passed on to a query modifier. The filter implements this selection as follows: if a query does not directly involve backlogs, but only non-invalidated, predefined views, it is passed on to the DBMS; if the query is from a non-invalidated application program, it is passed on to the DBMS; if the query is from the VS itself, it is passed on to the DBMS; finally, if the query is an interactive user query, it is passed on to the query modifier. An error occurs if the query is from an invalidated view or application program. Validation information for views and application programs is supplied by the VS (see below) and is stored in the data dictionary of the DBMS. Validated views and application programs are guaranteed not to access vacuumed data.

When a query, Q, arrives at the *query modifier*, its modification, $M_V Q$, is created (how this is done will be discussed later). Query $M_V Q$ gives the same result on $M_V \mathcal{DB}$ as does Q, that is $M_V Q \equiv_V Q$. Also $M_V Q$ avoids vacuumed data, that is $\forall \mathcal{DB}\ (M_V Q(\mathcal{DB}) = M_V Q(M_V \mathcal{DB}))$. A test procedure then tests Q and $M_V Q$[3]. If the test succeeds, Q does not reference vacuumed data and is passed on to the DBMS for processing. If the test fails, Q may reference vacuumed data, and it may be impossible to process without modifications. As a result, $M_V Q$ and possibly other similar queries that also avoid vacuumed data will be presented to the user. The user can choose any of the queries offered by the VS for processing, can modify any query expression and issue the result for processing, or can issue a completely new query. See figure 2.

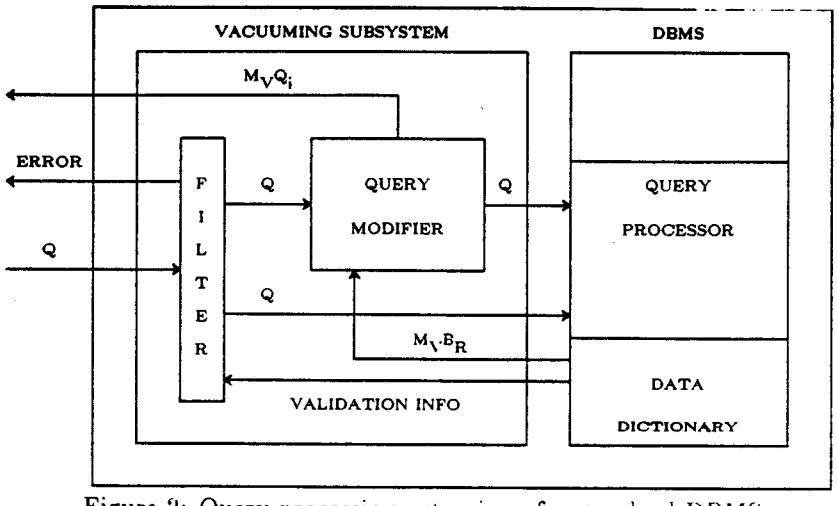

Figure 2: Query processing extension of a standard DBMS.

Together, the vacuuming filter and the query modifier guarantee that vacuuming will never change the recorded history. They do so by allowing only queries, Q, that if asked previously would have resulted in the same answer as if asked now:

$$\forall \mathcal{S}(\mathcal{DB})\ (\forall V\ (\forall I \in \mathcal{I}(\mathcal{DB})\ (Q(M_V \mathcal{DB}) = Q(\mathcal{DB})))) \tag{1}$$

[3]The general problem of determining equivalence of relational expressions is NP-complete, but efficient algorithms exist for determining equivalence for an important subset of expressions (most practical SPJ-queries) [4, 3, 20]. The query modifier's test procedure will never succeed if, in fact, $Q \not\equiv M_V Q$ (soundness), but it may fail to detect equivalence between complicated expressions (incompleteness). While a sound and complete procedure is preferable, the incompleteness is only a minor inconvenience in practice.

Specification of vacuuming is the responsibility of the DBA. To support the specification function, a VS should offer three tools: a validate tool, a help tool, and a warning tool. The validate tool is essential in that it ensures that specifications are correct (see below). The help and warning tools advise the DBA about what should be vacuumed and about which specifications seem inappropriate, respectively. These two tools are less essential.

When the vacuuming specification for a backlog relation is updated, *validity* is ensured by performing five checks.

1. The new vacuuming specification must be *growing*—a moving removal specification is not valid. Note that an initially valid specification can be rendered moving by updating not only the ρ part but also by updating the κ part.

2. There must be no conflicts between the vacuuming specification and *integrity constraints* (ICs). A conflict exists if the execution of a vacuuming specification would cause an IC to be violated. A conflict also exists if a vacuuming specification would result in removal of data needed to compute an IC. When conflicts arise, there is a choice to either modify the IC, modify the vacuuming specification, or modify both.

3. Possible conflicts between the specification and the set of current *view definitions* must be detected. A conflict exists if any view cannot be computed because of vacuuming. When a conflict arises, the data dictionary entries of the views involved are updated to signify that the views are invalidated, and the DBA is notified. Also, modified view definitions are made available to the owners of invalidated views. This is done to allow for easy conversion into validated views.

4. Possible conflicts between the specification and the queries of *application programs* must be detected so that no validated application programs attempt to access vacuumed data. In the case of a conflict, the data dictionary is again updated to reflect this, and modifications of the invalidated queries are made available for the owners of the involved programs.

5. The specification must not vacuum data needed by *(other) specifications* in order for them to serve as vacuuming specifications. An intra-specificational conflict is limited to the specification for a single backlog, and an inter-specificational conflict involves more specifications for more than one backlog. The DBA must resolve possible conflict.

The validate tool is activated when the set of vacuuming specifications, ICs, views, and application programs is updated. Conflict detection can be implemented using a modification-and-test procedure. For example, to test a view, its modification with respect to the current specification is constructed and is tested for equivalence with the original view. If the test fails, the view is invalidated; otherwise, it is validated. Figure 3 below illustrates the validate tool.

A *help tool* will have access to information about query patterns, about space consumption by the individual backlogs of the database, and about available storage. From this, space-consuming backlogs can be singled out together with regions of backlogs rarely referenced. Depending on the need for vacuuming—as indicated by the

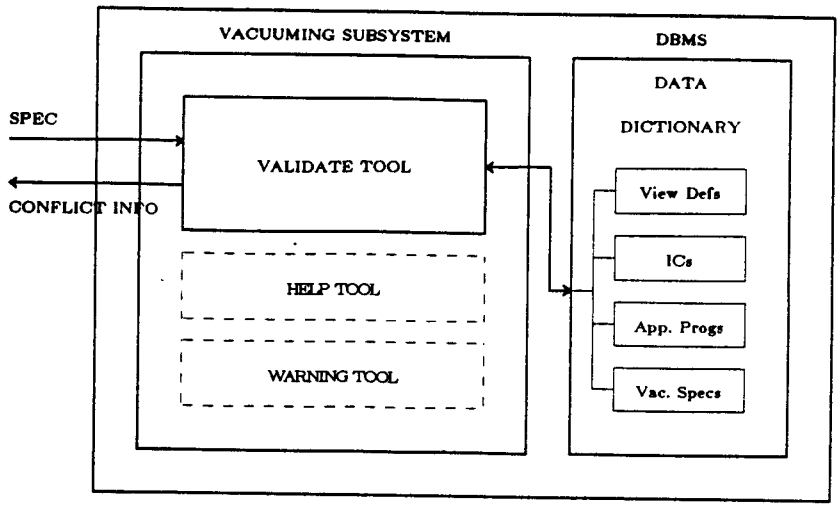

Figure 3: Functionality of the validate tool.

amount of available storage—the DBA can now specify vacuuming of appropriate portions of the least used and most space consuming data.

A *warning tool* tries to identify possible bad effects of the current specifications in general and of the updated specification in particular. As one example, we consider it generally imprudent to specify vacuuming of data that query patterns show are accessed frequently. By retaining information about how queries have been modified, it is also possible to point out trouble spots.

4 Taxonomy of Vacuuming Specifications

In this section, we present three classifications of vacuuming specifications. The first and second classifications are orthogonal and are both used for characterization of the classes of the third classification.

The first classification, A - D, makes use of two dimensions to distinguishes between specifications. Thus, specifications that are based on only the *Time* attribute are classified differently from those based on any set of attributes. In addition, specifications that remove complete tuples, and specifications that remove individual attribute values are classified differently. We have singled out the *Time* attribute because we consider it fundamental among attributes: the variable *NOW* takes as values elements of the domain of *Time*, tuples of backlogs are entered in time order, and we expect that vacuuming will often be specified in terms of *Time*. The second dimension is useful because removal of complete tuples is conceptually simpler to understand and technically much simpler than removal of individual attribute values.

CLASS A: `removal of complete tuples for time intervals` This class of specifications involve only selections on attribute *Time*. If present, projections are trivial in that they involve all the attributes of the backlog relation in question and can be deleted without changing the specification. Figure 4 illustrates specification V_1 of section 2.

CLASS B: `removal of complete tuples` This class generalizes the previous class by allowing selections on any

attribute. Implementations of the standard relational model typically support this class of deletions. Specification V_2 is an example. Specifications belonging to this class do not introduce vacuuming-null values.

CLASS C: **removal of all attribute values in time intervals** Specifications of this class can utilize the projection operator non-trivially. Thus, this class generalizes CLASS A. Selections can only use attribute *Time* as parameter. When a backlog vacuumed in this way is retrieved, it will appear with "blank" areas with special vacuuming-null values. Figure 5 visualizes a backlog vacuumed this way.

CLASS D: **removal of attribute values** This class generalizes CLASS B by allowing vacuuming of individual attributes, and it generalizes CLASS C by allowing selection to depend on any attribute. Figure 6 illustrates the kinds of patterns that can result from this kind of vacuuming.

Figure 7 visualizes the generalization structure of the classification A - D. The remaining parts of the figure will be explained next.

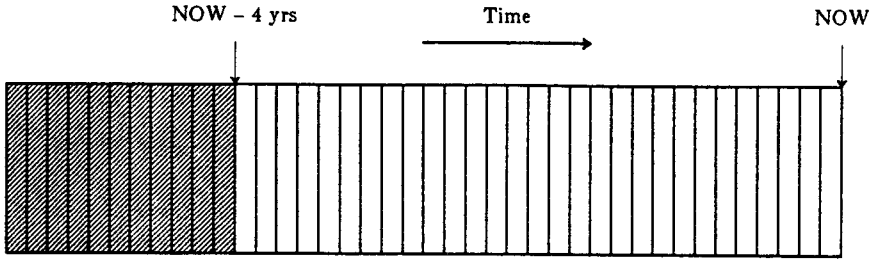

Figure 4: Visualization of tuple removal. The specification $\rho(B_{Emp}) : \sigma_{Time \leq NOW-4yrs} B_{Emp}$ is used.

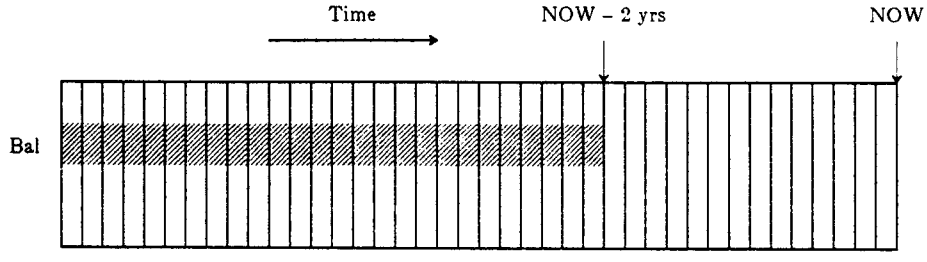

Figure 5: Visualization of attribute value removal for time periods. The figure corresponds to the specification: $\rho(B_{Emp}) : \pi_{Bal}\, \sigma_{Time<NOW-2yrs} B_{Emp}$.

To motivate the second classification, I - III, note that in DM/T we distinguish between backlog queries and time slice queries. While backlog queries apply directly standard relational algebra operators to time sliced backlogs, time slice queries involve only backlogs in the sense that they derive time sliced user-defined relations from these. In this classification, we distinguish between specifications that do or do not specify time slices of backlogs.

The classification is significant because, as we shall see later, time slice queries on backlogs vacuumed according to specifications of CLASS I may have modified queries that are problematic. The opposite, backlog queries on backlogs vacuumed according to specifications of CLASS II, is less problematic. Still, if backlog queries (time slice queries)

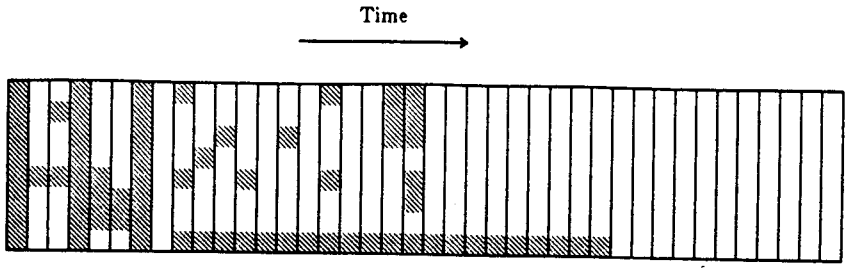

Figure 6: Visualization of general attribute value removal.

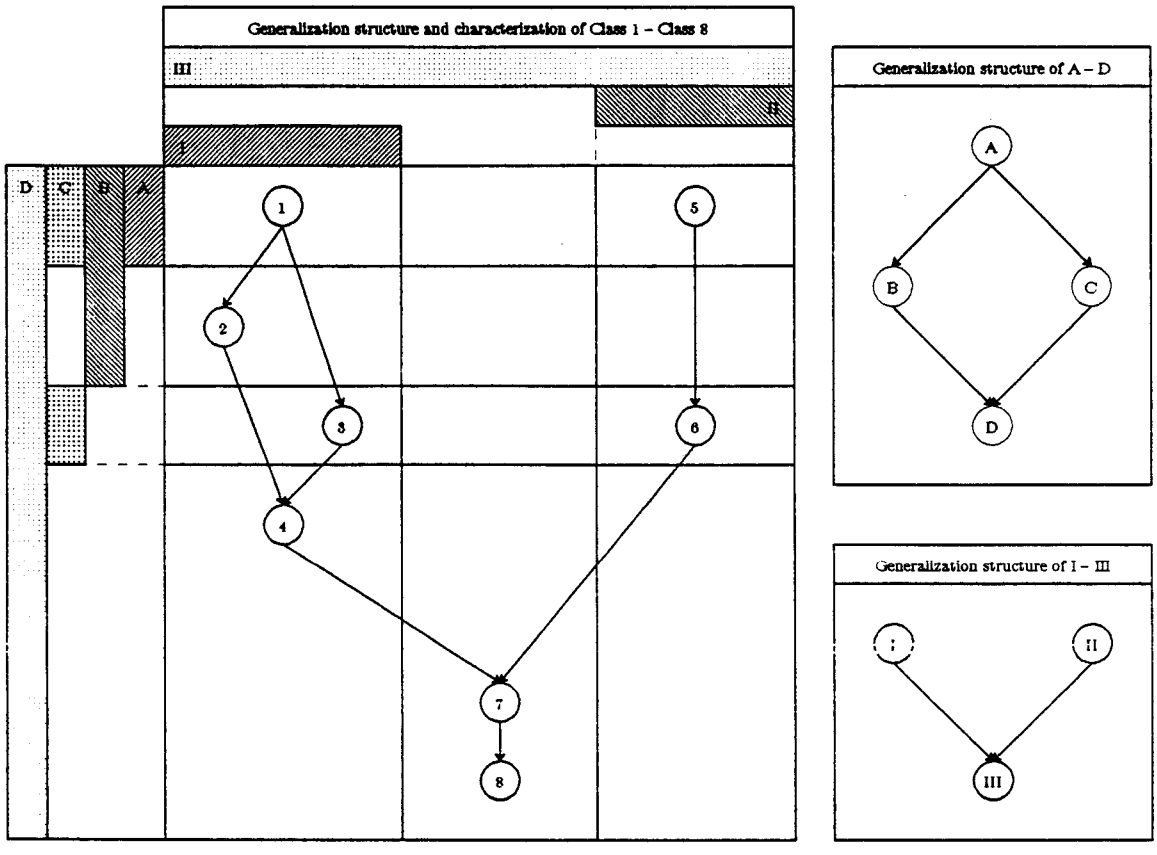

Figure 7: Characterization of classes CLASS 1 - CLASS 8.

are predominant on a given backlog, it is usually preferable to vacuum that backlog according to a specification of CLASS I (CLASS II).

CLASS I: **backlog based specifications** Specifications of this class contain only backlog-query expressions: no time sliced backlog expressions appear. All specifications in section 2 belong to this class.

CLASS II: **time slice based specifications** For a specification to belong to this class, each part of it must involve a time slice; however, it can (and typically will) also involve backlog-query expressions.

CLASS III: **mixed specifications** This category allows both time slice expressions and backlog-query expressions to be used in specifications, but neither time slice expressions nor backlog-query expressions are required.

The generalization structure of this classification is visualized in figure 7.

The query language can be used in many ways to specify data to be vacuumed. The third classification consists of 8 classes of queries, some very restricted and some very general. In choosing these, we have focused on conceptual simplicity, usefulness, and ease of implementation. The first classes are the simplest and easiest to understand, and the tendency, as we go down the list, is that they become more complex, but also more powerful. By presenting this classification and basing the subsequent presentation of vacuuming on it, we supply the designer of a VS with a valuable tool: we make it simple to choose a functionality (and complexity) that matches the particular requirements at hand.

Figure 7 illustrates the generalization structure of the classification 1 to 8, and it characterizes these classes in terms of the classes of the first (left columns) and second classification (top rows). For example, CLASS 2 generalizes CLASS 1 and is itself a specialization of CLASS 4. Also, any specification of CLASS 2 is an example of a specification of CLASS B and CLASS I. Figure 8 displays the syntax of the classes 1 through 8.

CLASS 1: **time-selection based vacuuming** This class is both very simple, fundamental, and useful. It removes tuples depending only on their time stamp attribute values. This class of specifications is represented by V_1 (section 2 and figure 4).

CLASS 2: **selection based vacuuming** This class allows for selections to involve not only attribute *Time*, but any attribute of B_R. The specification $\{V_1, V_2, V_3\}$ is of this type. Specifications in this class will not result in vacuuming-null values in query results (see figure 4).

CLASS 3: **projection based vacuuming** Elements of this class—where a projection of one or more attributes of a backlog is specified along with a selection on the attribute *Time*—all belong to CLASS C (see figures 8 and 5). Removal of attribute values as they get old is potentially very useful, and this class is the simplest for doing so. Note that columns of query results may contain vacuuming-null values when this kind of vacuuming is used.

CLASS 4: **general selection/projection based vacuuming** Generalizing all the previous ones, this class is contained in CLASS D (and not in classes B and C). Any SP query involving attributes of B_R is allowed. As

this, and consequently the previous three classes, is also contained in CLASS I, it is well suited for vacuuming backlogs that are mainly used in backlog queries (i.e., not in time slice queries)—see the example following this classification.

CLASS 5: `time slice selection based vacuuming` This is the simplest class for tuple removal that affects only time slice queries in simple and obvious ways, typically by removing all data before a certain date if they are not needed for time slice queries on states after that date. This class can be thought of as the time slice equivalent of CLASS 1.

CLASS 6: `time slice selection and projection based vacuuming` This is a simple generalization of the previous class where not only complete tuples, but also arbitrary projections can be removed.

CLASS 7: `general vacuuming involving one backlog` This type of specification can contain any query formulated in terms of selection, projection, and semi-join on a single backlog and time slices of that backlog. It generalizes classes 1 - 6, and it is the first not to be contained in either CLASS I or CLASS II.

CLASS 8: `general vacuuming` This class further generalizes the previous one by allowing any backlog and any time slice as argument in specifications.

EXAMPLE: Here, we illustrate that backlog based specifications can affect time slice queries in subtle and unpleasant ways. Assume the existence of the specification:

$$V_1 \qquad \rho(B_{Emp}) : \sigma_{Time \leq NOW-4yrs} B_{Emp}$$

The results of $Emp(NOW)$, before and after vacuuming according to V_1, may almost be identical. However, there is no guarantee, and there is no general, simple, and useful way to predict—from the specification and the time slice query issued—what has been left out due to vacuuming. For some applications, it is unsatisfactory to only know that employees can be missing from the result of $Emp(NOW)$ if they were not modified during the last four years.

Consequently, if time slice queries are frequent, specifications belonging to CLASS I (e.g., specifications of CLASS 1 and CLASS 2) should be used with caution. On the other hand, if backlog queries (i.e., queries without time slice) are predominant, these classes are likely to be useful.

Conversely, CLASS 5 and CLASS 6 are both designed to contain only specifications that result in useful and predictable results of time slice queries on modified backlogs. In addition, these specifications do not make backlog queries useless. Consider the specification:

$$V_4 \qquad \rho(B_{Emp}) : B_{Emp}(NOW - 2yrs) - B_{Emp}(NOW - 2yrs) \rhd \pi_{Time} Emp(NOW - 2yrs)$$

Backlog queries on data less than two years old are totally unaffected, and backlog queries on data more than two years will only retrieve few data. □

NUMBER	SYNTAX	COMMENTS
1	$\sigma_{F(Time)}B_R$	Only selections are possible, and they can only be parameterized by $B_R.Time$.
2	$\sigma_F B_R$	Only selections are possible, but any attribute of B_R may be used.
3	$\pi_A\sigma_{F(Time)}B_R$	A is a list of attributes from B_R and selections can only be parameterized by $B_R.Time$.
4	$\pi_A\sigma_F B_R$	A is any list of attributes from B_R, and F is any selection involving attributes of B_R,
5	$B_R(t) - B_R(t) \ltimes \pi_{Time}R(t)$	t is an expression, possibly involving the variable NOW, that evaluates to a value of the domain $TTime$.
6	$\pi_A(B_R(t) - B_R(t) \ltimes \pi_{Time}R(t))$	A is any attribute of B_R.
7	$Q(B_R)$	Any query using only B_R including time slices of B_R.
8	Q	Any query, possibly referencing other relations than B_R. The query can only use standard relational algebra operators and time slice.

Figure 8: Overview of classification of vacuuming specifications. The column SYNTAX describes Q so that a specification V for B_R is of the form $V \leftarrow \omega(B_R) : Q \ [\ ,\ \omega(B_R) : Q\ ]^*$ where $\omega \rightarrow \rho \mid \kappa$ and with the metasymbols "$\rightarrow$", "[", "]", "*", "|".

5 Theoretical and Conceptual Framework

In addition to physical removal of vacuumed data, a VS augments the functionality of a temporal DBMS in two areas: (1) specification of vacuuming, and (2) queries against backlogs that have been vacuumed. Here, we motivate and define general concepts related to these areas. We will make use of the concepts in the sections 6 - 10.

Conceptually, there is a substantial difference between vacuuming and subsequent query processing when specifications remain within the boundaries of CLASS B, where the unit of vacuuming is tuples, and when vacuuming specifications are outside CLASS B where the unit of vacuuming is occurrences of attribute values. In the former case, modified backlogs can be represented by relational algebra expressions; in the latter case, this is not possible (we will show an example shortly). Due to the differences, we divide the section into three parts. We first concentrate on concepts that are identical for both cases. Then we treat concepts that differ substantially when applied to vacuuming specifications within CLASS B and specifications outside of CLASS B.

Generally, a specification consists of a set of subspecifications—some ρ specifications and some κ specifications. Such specifications can *overlap*: part of data specified by one subspecification can be part of data specified by another subspecification. To obtain the largest degree of freedom of expression, we have chosen to allow overlaps between ρ specifications, between κ specifications, and between ρ and κ specifications. In the last case, κ specifications will override ρ specifications. This choice provides safety.

Vacuuming of backlogs can be described to the users in two ways: first, we can describe which data that remain (*positive form*); second, we can describe which data have gone (*negative form*). For some specifications, the positive form might be most easily understood by the user, and for others, the negative form might be easier to understand.

As a consequence of vacuuming, user-issued queries may be modified. Generalization and specialization are central relationships between queries and their modifications.

DEFINITION: Query Q_1 *generalizes* query Q_2, $Q_1 \succ Q_2$, iff $\forall I \in \mathcal{I}(\mathcal{DB})\ (Q_1(I) \supseteq Q_2(I))$. Similarly, query Q_1 *specializes* query Q_2 iff Q_2 generalizes Q_1. □

Generalization (and thus specialization) is reflexive, non-symmetric, and transitive.

Finally, it is a principle that when a vacuuming specification is changed, the new specification must generalize the previous specification. This ensures that what has already been vacuumed is part of what the new specification tells to vacuum.

We now consider specifications within CLASS B only. The concepts of growing and moving specifications are central. To motivate, note that by definition, and as a consequence of their very purpose, removals by specifications are irreversible. Also, note that specifications have the potential of changing over time, because the variable NOW can be used.

DEFINITION: Let $V \in$ CLASS B be a vacuuming specification for instances of $\mathcal{S}(B_R)$, and let x denote a tuple.

Specification V *grows* iff

$$\forall I \in \mathcal{I}(B_R)\ (\forall t \in [t_{init}; NOW]\ (\forall x\ ((x \in I(t) \wedge x \notin M_V I(t)) \Rightarrow \forall t' \geq t\ (x \notin M_V I(t')))))$$

□

The definition states that any tuple which has been removed from B_R by vacuuming according to V will never reappear in B_R—a tuple cannot be un-vacuumed. (Also, note that users cannot reinsert vacuumed tuples due to time-stamps and surrogates).

DEFINITION: Specification V *moves* iff

$$\exists I \in \mathcal{I}(B_R)\ (\exists t \in [t_{init}; NOW]\ (\exists t' \geq t\ (\exists x\ (x \in I(t) \wedge x \notin M_V I(t) \wedge x \in M_V I(t')))))$$

□

EXAMPLE: To illustrate the problem with moving specifications, consider the sample moving specification:

$$V_5 \qquad \rho(B_{Emp}) : \sigma_{NOW-2yrs<Time<NOW-1yrs} B_{Emp}$$

V_5 can be envisioned as a window that *moves* along B_{Emp} and deletes tuples as time passes. A VS could interpret the specification in two ways. First, it could simply delete tuples as they become more than one year old. This way—as data, previously between one and two years old, become more than two years old—data that were not supposed to be removed will have been deleted already, and the specification is destructive. Second, a VS could recognize the problem and not delete data that will be needed later. This way, nothing can be deleted, and the specification is useless. □

Specifications must be growing—moving specifications are invalid. However, note that a subspecification can be moving as long as the total specification is not (e.g., $\{V_1, V_2\}$). This is the case if tuples no longer in one part (V_2) are immediately included into another part (V_1).

Vacuuming concerns base data (i.e., the individual backlog relations). Here, we define how to describe a backlog modified by a specification in CLASS B using standard relational algebra.

DEFINITION: Let B_R be a backlog modified by a vacuuming specification, $V \in$ CLASS B, and let $V = \{V_1, V_2, \ldots, V_n\}$ where an element V_i is a removal specification if $1 \leq i \leq k \leq n$ and otherwise a keep specification. Also, let the query expression of V_i be termed Q_i, $1 \leq i \leq n$. We define the modified backlog $M_V B_R$ as follows

$$M_V B_R = B_R - \left(\bigcup_{i=1}^{k} Q_i - \bigcup_{i=k+1}^{n} Q_i \right) \tag{2}$$

□

The modified backlog is obtained by first taking the union of all Q_i, $1 \leq i \leq k$, then subtract the union of all Q_i, $k < i \leq n$ from these, and finally subtract this result from B_R.

The formula (2) has a set of delete terms (Q_i, $i = 1, 2, \ldots, k$) and a set of keep terms (Q_i, $i = k+1, k+2, \ldots, n$). The reason why the keep terms are needed is that they may overlap the delete terms. As we will illustrate in later sections, it is possible to discard the keep terms by eliminating the overlaps. Upon doing this, there is the option of expressing vacuuming either positively or negatively. Also, when we later restrict the types of specifications allowed, the general formula can be made more specific.

Moving on to specifications outside CLASS B, we redefine some concepts and new ones appear.

Again, we require specifications to be growing. The units of deletion are now attribute values, and we use these to define growing and moving specifications.

DEFINITION: Let v denote an occurrence of an attribute value. Then, a specification, V, for a backlog, B_R, is *growing* iff[4]

$$\forall I \in \mathcal{I}(B_R)\ (\forall t \in [t_{init}; NOW]\ (\forall v\ ((v \in I(t) \wedge v \notin M_V I(t)) \Rightarrow \forall t' > t\ (v \notin M_V I(t')))))$$

□

As before, if a specification is not growing, it is moving.

DEFINITION: Let v denote an occurrence of an attribute value. Then, a specification, V, for a backlog, B_R, is *moving* iff

$$\exists I \in \mathcal{I}(B_R)\ (\exists t \in [t_{init}; NOW]\ (\exists t' > t\ (\exists v\ (v \in I(t) \wedge v \notin M_V I(t) \wedge v \in M_V I(t')))))$$

□

For later use, we define the concepts of subtuple and supertuple.

DEFINITION: Let a relation R be given. If a tuple t_{SUB} of R can be obtained from another tuple t_{SUP} of R by substituting vacuuming-null values for zero or more of the attribute values of t_{SUP}, then t_{SUB} is said to be a *subtuple* of t_{SUP} and t_{SUP} a *supertuple* of t_{SUB}. □

For example, the tuple (10, -, 30k, -) is a subtuple of the tuple (10, M, 30k, 0), "-" being the vacuuming-null value.

In general, relational algebra is not sufficient to express a backlog modified by a specification not in CLASS B. The problem surfaces when a specification has several parts.

EXAMPLE: With the definitions of formula (2) in effect, define $R = Emp$, $n = k = 2$, $Q_1 = \pi_{Sex}\sigma_{Sal>30k}B_{Emp}$, and $Q_2 = \pi_{Bal}B_{Emp}$. This means that V_1 and V_2 specify removal from B_{Emp} of attribute value occurrences of Sex in all tuples with Sal value more than 30k and all occurrences of values of attribute Bal. This cannot be specified as $M_V B_{Emp} = B_{Emp} - (Q_1 \cup Q_2) = B_{Emp} - (\pi_{Sex}\sigma_{Sal>30k}B_{Emp} \cup \pi_{Bal}B_{Emp})$. Arguments of the union and difference operators must have compatible schemas. □

[4] We use "$v \in R$" to mean that attribute value occurrence v belongs to a tuple in R.

Consequently, we need another formalism for expressing a backlog that has been vacuumed. For this purpose, we introduce the concept of *tagged relation.*

DEFINITION: A tagged version, $\hat{R}$, of a relation, R, is the relation, R, where each attribute value may be tagged[5]. □

Next, we need to define how specifications outside CLASS B generate tagged versions of the backlogs, they are specifications for. We do this by letting the operators that appear in such specifications work like their tagged counterparts as they will be defined shortly. The tagged operators are also used for combining the tagged relations of subspecifications into other tagged relations. Finally, we need operators to transform tagged relations into ordinary relations.

EXAMPLE: The specification $\rho(B_{Emp}) : \pi_{Bal}B_{Emp}$ results in the tagged relation $\hat{B}_{Emp}$ defined as B_{Emp} with all occurrences of values of attribute Bal tagged. The result of the specification $\rho(B_{Emp}) : \pi_{Sex}\sigma_{Sal>30k}B_{Emp}$ is B_{Emp} defined as B_{Emp} with the occurrences of values of attribute Sex of the tuples $\sigma_{Sal>30k}B_{Emp}$ tagged. □

To define the tagged counterparts of the standard relational algebra operators, we first define a help predicate, tag, on attribute value occurrences of tagged relations.

DEFINITION: Let v be an occurrence of an attribute value in a tuple in a tagged relation.

$$tag(v) = \begin{cases} true & \text{if attribute value } v \text{ is tagged} \\ false & \text{if attribute value } v \text{ is not tagged} \end{cases} \tag{3}$$

□

A tagged relation, $\hat{R}$, where $\forall v \in \hat{R}\,(tag(v))$ is denoted $\hat{R}^{\bullet}$. and a tagged relation where $\forall v \in \hat{R}\,(\neg tag(v))$ is denoted $\hat{R}^{\circ}$.

We now define operators that manipulate tagged relations. These operators leave values unchanged: they only manipulate tags.

DEFINITION: Operator $\hat{\sigma}$ maps a tagged version of a relation into another tagged version of that relation. Let v be an occurrence of a value in the argument, and let $v_{\hat{\sigma}}$ be the corresponding occurrence in the result; also, let F be the selection criterion of $\hat{\sigma}$.

$$tag(v_{\hat{\sigma}}) = \begin{cases} true & \text{if } tag(v) \wedge \text{the tuple of } v \text{ is selected by } F \\ false & \text{otherwise} \end{cases} \tag{4}$$

Occurrences that are untagged are treated as null values by operator $\hat{\sigma}$.

Operator $\hat{\pi}$ also maps a tagged version of a relation into another tagged version of that relation. Again, let v be an occurrence of a value in the argument, and let $v_{\hat{\pi}}$ be the corresponding occurrence in the result; also, let A be the

[5] Let S be the schema of a database. Two tagged relation-valued query expressions, $\hat{Q}_1$ and $\hat{Q}_2$ are equivalent, $\hat{Q}_1 \hat{\equiv} \hat{Q}_2$, iff $\nu\hat{Q}_1 \equiv \nu\hat{Q}_2$ (Similarly, Q_1 generalizes Q_2, $Q_1 \succeq Q_2$, iff $\nu Q_1 \succ \nu Q_2$.). Operator ν will be defined shortly.

projection list of $\hat{\pi}$.

$$tag(v_{\hat{\pi}}) = \begin{cases} true & \text{if } tag(v) \wedge v \text{ belongs to an attribute in } A \\ false & \text{otherwise} \end{cases} \tag{5}$$

Next, we define an operator, $\hat{\cup}$. It can be applied to two tagged versions of the same relation. The result is another tagged version of that relation. Let v_{lhs} and v_{rhs} be the corresponding attribute values of the left and the right hand side arguments of operator $\hat{\cup}$, respectively. Then, the tag of the corresponding value of the result, $v_{\hat{\cup}}$, is defined by

$$tag(v_{\hat{\cup}}) = tag(v_{lhs}) \vee tag(v_{rhs}) \tag{6}$$

We define an operator $\hat{-}$. When applied to two tagged versions of the same relation, the result of $\hat{-}$ is another tagged version of the relation:

$$tag(v_{\hat{-}}) = tag(v_{lhs}) \wedge \neg tag(v_{rhs}) \tag{7}$$

Cartesian product for relations, $\times$, works for tagged relations as well. It simply leaves tags unchanged.
Semi-join for tagged relations, $\hat{\ltimes}_F$, is defined as follows:

$$\hat{R}_1 \hat{\ltimes}_F \hat{R}_2 = \pi_{Att(R_1)} \hat{\sigma}_F (\hat{R}_1 \times \hat{R}_2) \tag{8}$$

We use $\hat{\ltimes}$ without an index to denote equi-semijoin. We also define an operator, η, that reverses the tags of a tagged relation:

$$tag(v_\eta) = \neg tag(v) \tag{9}$$

□

Finally, we define operators that transform tagged relations into relations.

DEFINITION: When applied to a relation (lhs) and a tagged version of that relation (rhs), the result of operator $\dot{-}$ is the relation (lhs) modified as follows: first, each tuple, with all attribute values tagged in the rhs argument, is removed; second, each attribute value, tagged in the rhs argument, is substituted by a vacuuming-null value. We also define ν, an unary version of $\dot{-}$. It transforms a tagged relation into a relation by removing all tuples where each attribute value is tagged and by substituting each remaining tagged attribute value by a vacuuming-null value. □

We are now in a position to define vacuuming of a backlog modified by a specification in CLASS D, but not in CLASS B (if we are within CLASS B, tagged relations are completely irrelevant).

DEFINITION: With the assumptions listed prior to definition (2) in effect, except that V belongs to CLASS D, but not CLASS B, we define

$$M_V B_R = B_R \dot{-} \left(\widehat{\bigcup}_{i=1}^{k} Q_i \hat{-} \widehat{\bigcup}_{i=k+1}^{n} Q_i \right) = \nu \left(\widehat{\bigcup}_{i=1}^{k} Q_i \hat{-} \widehat{\bigcup}_{i=k+1}^{n} Q_i \right) \tag{10}$$

□

As in the case of CLASS B, overlaps between delete and keep terms can be eliminated, the keep terms can be discarded, and the remaining terms can be changed to obey either the positive or the negative form. Now, each of these forms can be transformed into two new fundamental forms. As the first option, each term can intuitively specify, for one or more attributes, which values to remove or not remove. As the second option, each term can intuitively specify, for a selection of tuples, the occurrences of which attribute values to remove or not remove. The negative forms of these we term *attribute normal form*, ANF$^-$, and *selection normal form*, SNF$^-$. Both of these tell what to vacuum. The corresponding positive forms, that tell what to retain, are termed ANF$^+$ and SNF$^+$, respectively. Thus, we have

$$\text{ANF}^-: \quad \nu\widehat{\bigcup}_{i=1}^{n} \pi_{A_i} Q_i \tag{11}$$

$$\text{ANF}^+: \quad \nu\eta\widehat{\bigcup}_{i=1}^{n} \pi_{A_i} Q_i \tag{12}$$

where A_i, $i = 1, 2, \ldots, n$, are disjoint subsets of the set of attributes of the backlog in question, and all Q_i are different (i.e., not equivalent). Thus, there is, at most, one term per attribute, and if attributes have the same Q_i, they are described by the same term. In (11), a term $\pi_{A_i} Q_i$ tells that for attributes A_i the occurrences of attribute values identified by Q_i *are not* parts of the modified backlog. In (12), a term $\pi_{A_i} Q_i$ tells that for attributes A_i the occurrences of attribute values identified by Q_i *are* parts of the modified backlog. The syntax of the forms SNF$^-$ and SNF$^+$ are identical to (11) and (12), respectively. However, now A_i, $i = 1, 2, \ldots, n$ are mutually distinct subsets of the set of attributes of the backlog in question. Thus, if the two terms i and j have identical subsets, $A_i = A_j$, then Q_i and Q_j are combined, and therefore A_i, $i = 1, 2, \ldots, n$, is the maximal subset for which Q_i is valid. Also the sets of tuples with tagged attribute value occurrences of the Q_i are mutually disjoint. The forms will be used extensively in the following sections.

For the above example motivating tagged relations, formula (10) becomes

$$M_V B_{Emp} \equiv B_{Emp} \dot{-} (\hat{\pi}_{Sex}\hat{\sigma}_{Sal>30k}\hat{B}^{\bullet}_{Emp} \hat{\cup} \hat{\pi}_{Bal}\hat{B}^{\bullet}_{Emp}) \quad [\text{ANF}^-] \tag{13}$$

$$\equiv \nu\eta(\hat{\pi}_{Bid,Op,Time,Id,Sal}\hat{B}^{\bullet}_{Emp} \hat{\cup} \hat{\pi}_{Sex}\hat{\sigma}_{Sal\leq 30k}\hat{B}^{\bullet}_{Emp}) \quad [\text{ANF}^+] \tag{14}$$

$$\equiv \nu(\hat{\pi}_{Sex,Bal}\hat{\sigma}_{Sal>30k}\hat{B}^{\bullet}_{Emp} \hat{\cup} \hat{\pi}_{Bal}\hat{\sigma}_{Sal\leq 30k}\hat{B}^{\bullet}_{Emp}) \quad [\text{SNF}^-] \tag{15}$$

$$\equiv \nu\eta(\hat{\pi}_{Bid,Op,Time,Id,Sal}\hat{\sigma}_{Sal>30k}\hat{B}^{\bullet}_{Emp} \hat{\cup} \hat{\pi}_{Bid,Op,Time,Id,Sal,Sex}\hat{\sigma}_{Sal\leq 30k}\hat{B}^{\bullet}_{Emp}) \quad [\text{SNF}^+] \tag{16}$$

For restricted classes of specifications, we can derive formulas more specific than (10). To do so, we need equivalence transformations for tagged relations. Such transformations can—in conjunction with transformations that allow to convert between ordinary relational algebra and tagged relational algebra—also prove beneficial when presenting to the users modified query expressions of queries against vacuumed backlogs because they can help simplify the modified queries.

THEOREM 5.1 *Transformations involving tagged relations. Let $\hat{R}_1$, $\hat{R}_2$, $\hat{R}_3$, $\hat{R}_4$, and $\hat{R}_5$ be tagged versions of the same relation, R, so that tagged values of the latter three belong to mutually disjoint tuples and $\hat{R}_3 \hat{\cup} \hat{R}_4 \hat{\cup} \hat{R}_5 \hat{\equiv} \hat{R}^{\bullet}$.*

Let $F = F_1 \wedge F_2$ be a selection criterion on R, let $Att(R)$ denote the set of attributes of R, and let A, B, and C be sets of attributes from R so that A and B are disjoint. Finally, let v be an arbitrary attribute value occurrence from R and let v_1, v_2, and v_3 be the tagged counterparts in $\hat{R}_1$, $\hat{R}_2$, and $\hat{R}_3$, respectively.

$$\sigma_F \nu\eta \hat{R}_1 \equiv \nu\eta\hat{\sigma}_F \hat{R}_1 \tag{17}$$

$$\pi_A \nu\eta \hat{R}_1 \equiv \nu\eta\hat{\pi}_A \hat{R}_1 \tag{18}$$

$$\hat{\sigma}_F \hat{R}_1 \mathrel{\hat{\equiv}} \hat{\sigma}_{F_1}\hat{\sigma}_{F_2}\hat{R}_1 \tag{19}$$

$$\hat{\sigma}_F(\hat{R}_1 \hat{\cup} \hat{R}_2) \mathrel{\hat{\prec}} \hat{\sigma}_F \hat{R}_1 \hat{\cup} \hat{\sigma}_F \hat{R}_2 \tag{20}$$

$$\hat{\sigma}_F(\hat{R}_1 \hat{\cup} \hat{R}_2) \mathrel{\hat{\equiv}} \hat{\sigma}_F \hat{R}_1 \hat{\cup} \hat{\sigma}_F \hat{R}_2 \quad [\hat{R}_1 \hat{\cup} \hat{R}_2 \text{ in SNF}^+ \text{ or SNF}^-] \tag{21}$$

$$\hat{\pi}_A\hat{\sigma}_{F_A}\hat{R}_1 \hat{\cup} \hat{\pi}_B\hat{\sigma}_{F_B}\hat{R}_1 \mathrel{\hat{\equiv}} \hat{\pi}_{A\cup B}\hat{\sigma}_{F_A \wedge F_B}\hat{R}_1 \hat{\cup} \hat{\pi}_A\hat{\sigma}_{F_A \wedge \neg F_B}\hat{R}_1 \hat{\cup} \hat{\pi}_B\hat{\sigma}_{F_B \wedge \neg F_A}\hat{R}_1 \tag{22}$$

$$\eta(\hat{\pi}_A \hat{R}_1 \hat{\cup} \hat{\pi}_B \hat{R}_2) \mathrel{\hat{\equiv}} \hat{\pi}_{Att(R)\setminus A\setminus B}(\hat{R}_1)^\bullet \hat{\cup} \hat{\pi}_A\eta\hat{R}_1 \hat{\cup} \hat{\pi}_B\eta\hat{R}_2 \tag{23}$$

$$\eta(\hat{\pi}_B \hat{R}_3 \hat{\cup} \hat{\pi}_C \hat{R}_4) \mathrel{\hat{\equiv}} \hat{\pi}_{Att(R)\setminus B}\hat{R}_3 \hat{\cup} \hat{\pi}_{Att(R)\setminus C}\hat{R}_4 \hat{\cup} \hat{\pi}_{Att(R)}\hat{R}_5 \tag{24}$$

$$\hat{\pi}_A(\hat{R}_1 \hat{\cup} \hat{R}_2) \mathrel{\hat{\equiv}} \hat{\pi}_A \hat{R}_1 \hat{\cup} \hat{\pi}_A \hat{R}_2 \quad \text{[distributivity of } \hat{\pi} \text{ over } \hat{\cup}] \tag{25}$$

$$\hat{\sigma}_F(\hat{R}_1 \hat{-} \hat{R}_2) \mathrel{\hat{\succ}} \hat{\sigma}_F \hat{R}_1 \hat{-} \hat{\sigma}_F \hat{R}_2 \tag{26}$$

$$\hat{\pi}_A(\hat{R}_1 \hat{-} \hat{R}_2) \mathrel{\hat{\equiv}} \hat{\pi}_A \hat{R}_1 \hat{-} \hat{\pi}_A \hat{R}_2 \quad \text{[distributivity of } \hat{\pi} \text{ over } \hat{-}] \tag{27}$$

$$\hat{R}_1 \hat{\cup} (\hat{R}_2 \hat{\cup} \hat{R}_3) \mathrel{\hat{\equiv}} (\hat{R}_1 \hat{\cup} \hat{R}_2) \hat{\cup} \hat{R}_3 \quad \text{[associativity of } \hat{\cup}] \tag{28}$$

$$\hat{R}_1 \hat{\cup} \hat{R}_2 \mathrel{\hat{\equiv}} \hat{R}_2 \hat{\cup} \hat{R}_1 \quad \text{[commutativity of } \hat{\cup}] \tag{29}$$

$$\hat{R}_1 \hat{-} (\hat{R}_2 \hat{\cup} \hat{R}_3) \mathrel{\hat{\equiv}} (\hat{R}_1 \hat{-} \hat{R}_2) \hat{-} \hat{R}_3 \tag{30}$$

$$\eta\hat{\pi}_A \hat{R}_1 \mathrel{\hat{\equiv}} \hat{\pi}_{Att(R)\setminus A}(\hat{R}_1)^\bullet \hat{\cup} \hat{\pi}_A\eta\hat{R}_1 \tag{31}$$

$$\eta\hat{\sigma}_F \hat{R}_1 \mathrel{\hat{\equiv}} \hat{\sigma}_{\neg F} \hat{R}_1 \tag{32}$$

$$\hat{\pi}_A\hat{\pi}_B \hat{R}_1 \mathrel{\hat{\equiv}} \hat{R}_1^\circ \tag{33}$$

$$\hat{\pi}_\emptyset \hat{R}_1^\bullet \mathrel{\hat{\equiv}} \hat{R}_1^\circ \tag{34}$$

$$\hat{R}_1^\circ \hat{\cup} \hat{R}_1 \mathrel{\hat{\equiv}} \hat{R}_1 \tag{35}$$

$$\hat{R}_1^\bullet \hat{\cup} \hat{R}_1 \mathrel{\hat{\equiv}} \hat{R}_1^\bullet \tag{36}$$

$$\hat{\sigma}_{False} \hat{R}_1 \mathrel{\hat{\equiv}} \hat{R}_1^\circ \tag{37}$$

$$\nu\hat{R}_1^\circ \equiv R \tag{38}$$

$$\nu\hat{R}_1^\bullet \equiv \emptyset_R \tag{39}$$

$$\eta\eta\hat{R}_1 \equiv \hat{R}_1 \tag{40}$$

PROOF:

(17): If v_1 is in the result of the lhs, then v_1 is in the result of the rhs. Also, if v_1 is in the result on the rhs, then v_1 is in the result on the lhs, because $\hat{\sigma}$ treats tagged attribute value occurrences as nulls (as does $\nu \circ \eta$).

(18): Note that the schema of the rhs result is generally a super schema of that of the lhs result. Also note, however,

that all the extra attributes of the rhs have solely vacuuming-null values. Even though the equivalence strictly speaking does not hold, we assume that a DBMS is able to simply remove superfluous attributes. When doing so, the equivalence follows immediately from the definition of $\hat{\pi}$. Similarly, we will feel free to use the equivalence: $\pi_A \nu\eta\hat{\pi}_B\hat{R}_1 \equiv \nu\eta\hat{\pi}_{A\cap B}\hat{R}_1$.

(19): Only attribute value occurrences in tuples selected by F_2 are tagged in $\hat{\sigma}_{F_2}$, and only occurrences tagged in $\hat{\sigma}_{F_2}$ and selected by F_1 are tagged in the rhs result. On the lhs, only occurrences in tuples selected by $F = F_1 \wedge F_2$ are tagged.

(20): If v is tagged on the rhs, then v is in a tuple selected by F, and v is tagged in either $\hat{R}_1$ or $\hat{R}_2$. Thus, v is tagged in $\hat{R}_1\hat{\cup}\hat{R}_2$ and is therefore tagged on the lhs. The equivalence does not hold true because $\hat{\cup}$ combines tags so that, for example, a selection $Att_1 = Att_2$ may tag the occurrences in a tuple on the lhs and not on the rhs.

(21): The condition means that the set of tuples with tagged occurrences in R_1 and the set of tuples with tagged occurrences in R_2 are disjoint. Thus, tagged occurrences in tuples selected on the lhs will also be tagged on the rhs (the selection of one tuple does not depend on the tagging in other tuples).

(22): If an occurrence is tagged by the first term on the lhs, it is tagged by the first or second term on the rhs (depending on whether the tuple it is in satisfies F_B or not). If an occurrence is tagged by the second term on the lhs, it is tagged by the first or third term on the rhs. If an occurrence is tagged by the first term on the rhs, it is tagged by the first or second term on the lhs (depending on whether it belongs to an attribute in A or B). Occurrences tagged by the second and third terms on the rhs are tagged by the first and second terms on the lhs, respectively.

(23): An occurrence tagged by the first term on the lhs is not tagged by the first term (disjoint projections), the second term (reversely tagged argument to the projection), and the third term (disjoint projections) on the rhs. Due to symmetry, an occurrence tagged by the second term on the lhs is also untagged on the rhs. An occurrence tagged by the first term on the rhs is not tagged on the lhs (disjoint projections), and an occurrence tagged by the second term on the rhs is not tagged by the first term on the lhs (reversely tagged argument to the projection) and the second term on the lhs (disjoint projections). Again, due to symmetry with the case of the second term, an occurence tagged by the third term on the rhs is untagged on the lhs.

(24): An occurrence tagged by the first term on the lhs is not tagged by the first term (disjoint projections), the second term (disjoint selection), and the third term (disjoint selection) on the rhs. The second term on the lhs is treated symmetrically. Occurrences tagged on the rhs are untagged on the lhs due to disjoint projections and selections.

(25): Trivial.

(26): Assume that v is tagged in the lhs result. Then, $tuple(v)$ must be selected by $\hat{\sigma}_F$ in $(\hat{R}_1\hat{-}\hat{R}_2)$, and $\neg tag(v_2)$. The former implies that $\hat{\sigma}_F$ selects $tuple(v)$ in $\hat{R}_1$. The latter implies that v is not tagged in $\hat{\sigma}_F\hat{R}_2$. Consequently, v is tagged in the rhs result.

(27): Projection leaves tags unchanged for occurrences that are projected, and thus the equivalence holds true in this case. For occurrences not projected, it removes tags. Therefore, occurences on the lhs will be untagged. Because

$False \wedge \neg False \equiv False$, occurrences on the rhs will be untagged, too.

(28) - (29): Trivial from the definition of $\hat{\cup}$.

(30): The equivalence follows from the definitions of $\hat{-}$ and $\hat{\cup}$, and from $x \wedge \neg(y \vee z) \equiv (x \wedge \neg y) \wedge \neg z$.

(31): An occurrence is tagged on the lhs if it is untagged in $\hat{R}_1$ or does not belong to an attribute in A. On the rhs, an occurrence is tagged if it belongs to an attribute outside of A, or if it belongs to an attribute in A and is untagged in $\hat{R}_1$.

(32) - (40): These observations follow trivially from the definitions. □

After having seen some of the central properties of the new operators, we can look back and justify their definitions. The equivalences (17) and (18) state that there is a very close correspondence between operators $\hat{\sigma}$, $\hat{\pi}$, and their standard relational algebra counterparts. Operators $\hat{\cup}$ and $\hat{-}$ do not have counterparts in relational algebra—it was exactly this deficiency that motivated tagged relational algebra. Operator $\hat{\cup}$ is designed for collecting tags of different ρ and κ expressions into single expressions reflecting all deletions and all data to keep, respectively. Operator $\hat{-}$ was shaped to combine ρ and κ specifications. Its definition reflects directly the semantics chosen for overlapping ρ and κ specifications, namely that κ specifications override ρ specifications.

Recall that equivalences as in the theorem above are utilized to express vacuumed backlogs. Transformation of a query, Q, against the original backlogs into a query, $M_V Q$, against the modified backlogs is done by substituting the original backlog name occurrences in Q by the expressions for the modified backlogs and applying the equivalences. Using only equivalences guarantee that the result of Q on the vacuumed backlogs and the result of $M_V Q$ on the original backlogs are the same. The approach ensures also that modified queries do not reference vacuumed data (i.e., they obey the law (1))—see section 3. Recall that a modified query expression is displayed to the user for confirmation before being processed. While still obeying (1), it is possible to obtain additional modified expressions perhaps more useful to the user by deviating slightly from the transformations based on the equivalences. Specifically, the technique for doing so is simply to remove conditions in queries that rely on vacuumed data. This results in more elaborate queries that may prove more informative. In section 7, we will demonstrate this.

This concludes the abstract treatment of vacuuming concepts. When we, in the following 5 sections, consider CLASS 1 to CLASS 8, we will make specific application of these general concepts. The distribution of the 8 classes is as follows:

	CLASS 1	CLASS 2	CLASS 3	CLASS 4	CLASS 5	CLASS 6	CLASS 7	CLASS 8
Section 6	×	×						
Section 7			×					
Section 8				×				
Section 9					×	×		
Section 10							×	×

The sections are, with minor variations, all organized the same way: The class(es) of specification(s) is (are) introduced, the syntax is explained, and the use of the particular class of specifications is illustrated by the way

of a sample specification. Then, issues pertinent to the design and update of specifications are discussed (e.g. growing/moving, conflicts, etc.). The problem of expressing vacuumed backlogs for the particular class in question is addressed; a formula is constructed (a special case of the general ones), and its use is demonstrated by means of the sample specification introduced previously. The issues of querying vacuumed backlogs are considered; the procedure for modifying a query on a vacuumed backlog is applied to a sample query against the backlog modified previously. Finally, we set forward some general observations on the relations between queries and modified queries.

6 Selection Based Vacuuming

The syntax of vacuuming expressions for CLASS 1 and CLASS 2 is $\sigma_{F(Time)}B_R$ and $\sigma_F B_R$, respectively (figure 8).

Specifications of both classes delete complete tuples, and because F can only be parameterized by $Time$, specifications of CLASS 1 can be visualized as removing time intervals of tuples; tuples deleted by a CLASS 2 specification lack this simple pattern because removal generally depends on multiple attributes.

Consider these specifications (from section 2):

$$V_1 \quad \rho(B_{Emp}) : \sigma_{Time \leq NOW-4yrs} B_{Emp}$$

$$V_2 \quad \rho(B_{Emp}) : \sigma_{NOW-4yrs < Time \leq NOW-2yrs \wedge Sex=F} B_{Emp}$$

$$V_3 \quad \kappa(B_{Emp}) : \sigma_{Op=D} B_{Emp}$$

Specification V_1 is in CLASS 1; V_2, V_3, and any combination of the specifications are in CLASS 2.

When updating a specification, several problems can occur. First, it does not make sense to change a specification so that it does not specify removal of data already lost due to vacuuming—an updated vacuuming specification must generalize its predecessor.

EXAMPLE: Assume that on May 11, 1990, 2:00 p.m. specification V_1 for B_{Emp} is in effect, and that a decision is made to retain data until they become 5 years old. It is illegal to just change $NOW - 4yrs$ of V_1 to $NOW - 5yrs$. Instead, we can substitute V_1 for this specification:

$$V_1' \quad \rho(B_{Emp}) : \sigma_{Time \geq May11,1990,2:00p.m.-4yrs} B_{Emp}$$

$$V_1'' \quad \rho(B_{Emp}) : \sigma_{Time \geq NOW-5yrs} B_{Emp}$$

Then, after May 11, 1991, 2:00 p.m., $V_1'' \succ V_1'$ and V_1' can be deleted. □

As another problem area, moving specifications must be avoided. This was dealt with in the previous section.

In section 2, we exemplified how a modified backlog (and a modified query), $M_V B_R$, could be derived from specification V and backlog B_R. Here, we derive two general formulas, valid for both CLASS 1 and CLASS 2. If $V = \{V_1, V_2, \ldots, V_n\}$, $V_i = \rho(B_R) : \sigma_{F_i} B_R$, $1 \leq i \leq k$, and $V_i = \kappa(B_R) : \sigma_{F_i} B_R$, $k+1 \leq i \leq n$, then we have

$$M_V B_R \equiv \sigma_{\wedge_{i=1}^{k} \neg F_i \vee (\vee_{i=k+1}^{n} F_i)} B_R \tag{41}$$

$$M_V B_R \equiv B_R - \sigma_{\vee_{i=1}^{k} F_i \wedge (\wedge_{i=k+1}^{n} \neg F_i)} B_R \tag{42}$$

Formula (41) tells what to retain, and formula (42) tells what to delete. They are derived from the general formula as follows:

$$\begin{aligned}
M_V B_R &= B_R - \left(\bigcup_{i=1}^{k} Q_i - \bigcup_{i=k+1}^{n} Q_i \right) \\
&\equiv B_R - \left(\bigcup_{i=1}^{k} \sigma_{F_i} B_R - \bigcup_{i=k+1}^{n} \sigma_{F_i} B_R \right) \\
&\equiv B_R - \left(\sigma_{\vee_{i=1}^{k} F_i} B_R - \sigma_{\vee_{i=k+1}^{n} F_i} B_R \right) \\
&\equiv B_R - \sigma_{\vee_{i=1}^{k} F_i \wedge \neg(\vee_{i=k+1}^{n} F_i)} B_R \\
&\equiv B_R - \sigma_{\vee_{i=1}^{k} F_i \wedge (\wedge_{i=k+1}^{n} \neg F_i)} B_R \\
&\equiv \sigma_{\neg(\vee_{i=1}^{k} F_i \wedge (\wedge_{i=k+1}^{n} \neg F_i))} B_R \\
&\equiv \sigma_{\wedge_{i=1}^{k} \neg F_i \vee (\vee_{i=k+1}^{n} F_i)} B_R
\end{aligned}$$

Here, we have utilized the equivalences: $\sigma_{F_1} R \cup \sigma_{F_2} R \equiv \sigma_{F_1 \vee F_2} R$, $\sigma_{F_1} R - \sigma_{F_2} R \equiv \sigma_{F_1 \wedge \neg F_2} R$, $R - \sigma_F R \equiv \sigma_{\neg F} R$, $\neg(F_1 \vee F_2) \equiv \neg F_1 \wedge \neg F_2$, and $\neg(F_1 \wedge F_2) \equiv \neg F_1 \vee \neg F_2$.

Any query, $Q(B_R)$, against B_R is transformed into $M_V Q = Q(M_V B_R)$, and if the test of the query modifier fails, then the modified query is presented to the user who then decides whether it should be processed right away or perhaps modified first.

EXAMPLE: If $Q = \sigma_F B_R$, then $M_V Q = \sigma_F M_V B_R = \sigma_{F \wedge F'} B_R = \sigma_{F \wedge [(\wedge_{i=1}^{j} \neg F_i) \vee (\vee_{i=j+1}^{n} F_i)]} B_R$. Similarly, if $Q = \pi_A B_R$, then $M_V Q = \pi_A \sigma_{F'} B_R$; if $Q = B_R \bowtie B_R$, then $M_V Q = \sigma_{F'} B_R \bowtie \sigma_{F'} B_R$. □

For an illustration of the modification of a user query on a modified backlog, see section 2. The extension of a transformed query $M_V Q$ of a query Q against a backlog relation that has been subject to vacuuming based on a specification of CLASS 1 and CLASS 2 fits the schema of Q, and no vacuuming-null values are introduced because only whole tuples are removed. Also, observe that a transformed query, $M_V Q$, specializes the original query, Q, so that $Q \succ M_V Q$.

7 Projection Based Vacuuming

In contrast to selection based vacuuming specifications which result in removal of complete tuples from a backlog, projection based vacuuming (CLASS 3) results in time-parameterized removal of all values of one or more attributes of a backlog. The syntax of vacuuming expressions of this class is $\pi_A \sigma_{F(Time)} B_R$ where A is a list of attributes, and $\sigma_{F(Time)}$ is a selection involving only attribute *Time*.

With this kind of vacuuming, all values of one or more attributes of tuples, where the value of attribute *Time* is in a specified interval, can be removed. This can result in "blank" areas of vacuuming-null values in results (figure 5). We stress that this class of vacuuming does not alter the schema of argument backlogs; at any time, a tuple of a relation $M_V B_R$ is a subtuple of a tuple of B_R.

Projection based vacuuming is very useful. For example, when tuples get old, we may not be interested in the values of all the attributes:

$$V_6 \qquad \rho(B_{Emp}) : \pi_{Sex,Sal}\sigma_{Time \leq NOW-2yrs} B_{Emp}$$

This specification deletes values of attributes *Sex* and *Sal* of tuples where the value of attribute *Time* is less than $NOW - 2yrs$. Following is a more elaborate projection based specification:

$$V_7 \qquad \rho(B_{Emp}) : \pi_{Bal,Sal,Sex}\sigma_{Time \leq NOW-6yrs} B_{Emp}$$

$$V_8 \qquad \rho(B_{Emp}) : \pi_{Bal,Sal}\sigma_{NOW-6yrs<Time \leq NOW-4yrs} B_{Emp}$$

$$V_9 \qquad \rho(B_{Emp}) : \pi_{Sal}\sigma_{NOW-4yrs<Time \leq NOW-2yrs} B_{Emp}$$

$$V_{10} \qquad \kappa(B_{Emp}) : \pi_{Id,Bal}\sigma_{NOW-5yrs<Time} B_{Emp}$$

Here, the time dimension is divided into five intervals, each with vacuuming of different sets of attribute values.

For a projection based specification to be legal, we require it to be growing, as in the case of selection based specifications. This issue was dealt with in section 5.

As indicated already, the convenient functionality of a specifications of CLASS 3—and of any specification, not in CLASS B—does not come for free: the introduction of tagged relations is necessary.

To express a backlog modified according to CLASS 3, we start out with the general formula, (10). There, the terms are simply denoted Q_i. Here, we can be more specific. First, observe that the deletion part of a specification of CLASS 3 *splits the backlog into time intervals with different sets of attributes with vacuumed values*, and note that such a set of attributes of an older interval is a superset of that of a newer. Second, identify this sequence of intervals with differing sets of missing values ($I_1, I_2, \ldots, I_n$ where $i \neq j \Rightarrow I_i \cap I_j = \emptyset$, and $\cup_{i=1}^{n} I_i = [t_{init}; NOW]$; also, denote the set of attributes of I_i with deleted values by A_i^- so that $i < j \Rightarrow A_j^- \subseteq A_i^-$.). Third, for the "keep" part, denote the corresponding intervals $J_1, J_2, \ldots, J_m$, and denote the sets of attributes with values that must be kept $B_1^+, B_2^+, \ldots, B_m^+$. Now, the expression of the modified backlog is given as[6]

$$M_V B_R \equiv B_R \stackrel{\bullet}{-} \underbrace{\left(\overbrace{\widehat{\bigcup}_{i=1}^{n} \hat{\pi}_{A_i^-} \hat{\sigma}_{I_i} \hat{B}_R^{\bullet}}^{\text{delete}} \stackrel{\frown}{-} \overbrace{\widehat{\bigcup}_{j=1}^{m} \hat{\pi}_{B_j^+} \hat{\sigma}_{J_j} \hat{B}_R^{\bullet}}^{\text{keep}} \right)}_{\text{net deletion}} \tag{43}$$

Before we simplify this formula further, we will construct an equivalent but different expression. Now, we observe that, due to ρ specifications, each attribute will have vacuuming-null values from some point in time and backwards. For each attribute, A_i, $1 \leq i \leq a$, there will exist a term $\hat{\pi}_{A_i} \hat{\sigma}_{t_{init} \leq Time \leq t_i^{\rho}} \hat{B}_R^{\bullet}$. Also, we observe, now due to κ specifications, that for each attribute, deletion is not permitted until after a certain point in time. Thus, for each

[6] This may include superfluous delete and keep terms with nothing to delete/keep.

attribute, there will exist a term $\hat{\pi}_{A_i}\hat{\sigma}_{t_i^\kappa \leq Time \leq NOW}\hat{B}_R^\bullet$. This gives an expression as follows:

$$M_V B_R \equiv B_R \stackrel{\bullet}{-} \underbrace{\left(\overbrace{\widehat{\bigcup}_{i=1}^{a} \hat{\pi}_{A_i}\hat{\sigma}_{t_{init} \leq Time \leq t_i^\rho}\hat{B}_R^\bullet}^{\text{delete}} \hat{-} \overbrace{\widehat{\bigcup}_{i=1}^{a} \hat{\pi}_{A_i}\hat{\sigma}_{t_i^\kappa \leq Time \leq NOW}\hat{B}_R^\bullet}^{\text{keep}} \right)}_{\text{net deletion}} \tag{44}$$

In order to simplify (43) and (44), observe that the only reason why we cannot discard the keep part is that occurrences of attribute values tagged in the delete part may be tagged in the keep part as well: there may be overlaps. Once the overlaps are eliminated, the delete parts may be discarded (i.e., $\hat{R}_1 \hat{-} \hat{R}_2 \hat{\equiv} \hat{R}_1$ if there is no overlap between $\hat{R}_1$ and $\hat{R}_2$). To eliminate overlaps, we use this equivalence:

$$\hat{\pi}_{A_1}\hat{\sigma}_{a \leq Time \leq b}\hat{R}^\bullet \hat{-} \hat{\pi}_{A_2}\hat{\sigma}_{c \leq Time \leq d}\hat{R}^\bullet \hat{\equiv} \tag{45}$$

$$(\hat{\pi}_{A_1}\hat{\sigma}_{(a \leq Time \leq b)\setminus(c \leq Time \leq d)}\hat{R}^\bullet \hat{\cup} \hat{\pi}_{A_1 \setminus A_2}\hat{\sigma}_{(a \leq Time \leq b)\cap(c \leq Time \leq d)}\hat{R}^\bullet) \hat{-} \hat{\pi}_{A_2}\hat{\sigma}_{c \leq Time \leq d}\hat{R}^\bullet$$

where there is no overlap between the keep and delete parts of the right hand side.

By means of repeated applications of this equivalence transformation, all overlaps can be eliminated in both formulas. Doing this gives new intervals. In formula (43), denote by K_k, $1 \leq k \leq p$, the resulting intervals that partition $[t_{init}; NOW]$ and have distinct sets of attributes with vacuumed values (denoted $C_1^-, C_2^-, C_3^-, \ldots, C_p^-$). The formula then becomes

$$M_V B_R \equiv B_R \stackrel{\bullet}{-} \underbrace{\widehat{\bigcup}_{k=1}^{p} \hat{\pi}_{C_k^-}\hat{\sigma}_{K_k}\hat{B}_R^\bullet}_{\text{net deletion}} \tag{46}$$

Here, each term tells to remove occurrences of values of some set of attributes for a selection, and the formula is thus in SNF$^-$.

To simplify formula (44), the values t_i^ρ and t_i^κ, $1 \leq i \leq a$, are compared, and $t_i = \min(t_i^\kappa, t_i^\rho)$. Then

$$M_V B_R \equiv B_R \stackrel{\bullet}{-} \underbrace{\widehat{\bigcup}_{i=1}^{a} \hat{\pi}_{A_i}\hat{\sigma}_{t_{init} \leq Time \leq t_i}\hat{B}_R^\bullet}_{\text{net deletion}} \tag{47}$$

In this formula, each attribute with values to vacuum is included in precisely one term that tells which occurrences of values for the attribute to remove: it is in ANF$^-$.

EXAMPLE: To illustrate, assume that B_{Emp} has been vacuumed according to $V = \{V_7, V_8, V_9, V_{10}\}$. First, we construct the SNF$^-$ formula. For deletions, we have $n = 4$ with $I_1 = [t_{init}; NOW - 6yrs]$, $A_1^- = Bal, Sal, Sex$; $I_2 =]NOW - 6yrs; NOW - 4yrs]$, $A_2^- = Bal, Sal$; $I_3 =]NOW - 4yrs; NOW - 2yrs]$, $A_3^- = Sal$; $I_4 =]NOW - 2yrs; NOW]$, $A_4^- = \emptyset$. For κ-specifications, we have $m = 2$ with $J_1 = [t_{init}; NOW - 5yrs[$, $B_1^+ = \emptyset$; $J_2 = [NOW - 5yrs; NOW]$, $B_2^+ = Id, Sal$. Upon having eliminated overlaps, we get $K_1 = I_1$, $C_1^- = A_1^-$; $K_2 =]NOW - 6yrs; NOW - 5yrs]$, $C_2^- = A_2^-$; $K_3 =]NOW - 5yrs; NOW - 2yrs]$, $C_3^- = A_3^-$; $K_4 = I_4$, $C_4^- = A_4^-$. The modified backlog is then given in SNF$^-$ as

$$M_V B_{Emp} \equiv B_{Emp} \stackrel{\bullet}{-} (\hat{\pi}_{Bal,Sal,Sex}\hat{\sigma}_{Time \leq NOW - 6yrs}\hat{B}_{Emp}^\bullet$$

$$\hat{\cup}\ \hat{\pi}_{Bal,Sal}\hat{\sigma}_{NOW-6yrs<Time\leq NOW-5yrs}\hat{B}^{\bullet}_{Emp} \qquad (48)$$
$$\hat{\cup}\ \hat{\pi}_{Sal}\hat{\sigma}_{NOW-5yrs<Time\leq NOW-2yrs}\hat{B}^{\bullet}_{Emp})$$

Now we proceed to derive the ANF$^-$ formula. Consider this table:

Attribute	ρ	κ	net ρ
Sal	$\rightarrow NOW-2yrs$	$NOW \leftarrow$	$\rightarrow NOW-2yrs$
Bal	$\rightarrow NOW-4yrs$	$NOW-5yrs \leftarrow$	$\rightarrow NOW-5yrs$
Sex	$\rightarrow NOW-6yrs$	$NOW \leftarrow$	$\rightarrow NOW-6yrs$
Id	$\rightarrow t_{init}$	$NOW-5yrs \leftarrow$	$\rightarrow t_{init}$
Time, Op, Bid	$\rightarrow t_{init}$	$NOW \leftarrow$	$\rightarrow t_{init}$

The second column illustrates delete terms: there are 3 non-trivial terms, one for each of the attributes *Sal*, *Bal*, and *Sex*. The third illustrates keep terms: there are non-trivial terms for attributes *Sal* and *Id*. The fourth column illustrate net deletions. Thus, we get the ANF$^-$ formula:

$$M_V B_{Emp} \equiv B_{Emp} \stackrel{\bullet}{-} (\hat{\pi}_{Sal}\hat{\sigma}_{Time\leq NOW-2yrs}\hat{B}^{\bullet}_{Emp} \hat{\cup}\ \hat{\pi}_{Bal}\hat{\sigma}_{Time\leq NOW-5yrs}\hat{B}^{\bullet}_{Emp}$$
$$\hat{\cup}\ \hat{\pi}_{Sex}\hat{\sigma}_{Time\leq NOW-6yrs}\hat{B}^{\bullet}_{Emp}) \qquad (49)$$

□

In the example, we illustrated an SNF$^-$ and an ANF$^-$ version of the modified backlog. Both of these tell what to remove. Using the operator η, we can also express a modified backlog by means of terms that show what not to delete. To get a version in SNF$^+$, the set of attributes of each term is replaced by its complement with respect to the set of all attributes of the backlog vacuumed, and η is applied to the union of the terms. To get a version in ANF$^+$, the selection for each term is replaced by its complement w.r.t. the interval $[t_{init}; NOW]$, and, again, the operator η is applied to the union of the terms.

We illustrate querying of a vacuumed backlog by an example.

EXAMPLE: Assume that we issue this query:

$$Q = \pi_{Id,Bal,Sex}\sigma_{NOW-7yrs\leq Time\leq NOW-1yrs\wedge Sal\geq 30k\wedge Sex=F}B_{Emp}$$

To get $M_V Q$, we substitute a definition of $M_V B_{Emp}$ for B_{Emp}. Using (48) and only equivalence transformations, we get

$$M_V Q \equiv B_{Emp} \stackrel{\bullet}{-} (\hat{\pi}_{Bid,Op,Time,Id,Sal,Bal,Sex}\hat{\sigma}_{Time\leq NOW-2yrs\vee Time>NOW-1yrs}\hat{B}^{\bullet}_{Emp}$$
$$\hat{\cup}\ \hat{\pi}_{Bid,Op,Time,Sal}\hat{\sigma}_{NOW-2yrs<Time\leq NOW-1yrs\wedge Sal<30k\wedge Sex=M}\hat{B}^{\bullet}_{Emp})$$

In K_1, nothing is selected as the selections on both *Sex* and *Sal* fail, and in K_2 and K_3, nothing is selected because the selection on *Sal* fails. To obtain a more elaborate, alternative expression, we use the technique of removing conditions that cannot be evaluated (or fail!) due to vacuuming.

$$M_V Q' \equiv B_{Emp} \stackrel{\bullet}{-} (\hat{\pi}_{Bid,Op,Time,Id,Sal,Bal,Sex}\hat{\sigma}_{Time<NOW-7yrs\vee Time>NOW-1yrs}\hat{B}^{\bullet}_{Emp}$$

$\hat{\cup}\ \hat{\pi}_{Bid,Op,Time,Sal,Bal,Sex}\hat{\sigma}_{NOW-7yrs\leq Time\leq NOW-6yrs}\hat{B}^{\bullet}_{Emp}$

$\hat{\cup}\ \hat{\pi}_{Bid,Op,Time,Sal,Bal}\hat{\sigma}_{NOW-6yrs<Time\leq NOW-5yrs\wedge Sex=M}\hat{B}^{\bullet}_{Emp}$

$\hat{\cup}\ \hat{\pi}_{Bid,Op,Time,Sal}\hat{\sigma}_{(NOW-5yrs<Time\leq NOW-2yrs\wedge Sex=M)\vee(NOW-2yrs<Time\leq NOW-1yrs\wedge Sex=M\wedge Sal<30k)}\hat{B}^{\bullet}_{Emp})$

In K_1, the restrictions on *Sal* and *Sex* are ignored, and the result will have vacuuming-null values for attributes *Bal* and *Sal*. In K_2, the restriction on *Sal* is ignored, and the attribute *Bal* will have vacuuming-null values in the result. In K_3, the restriction on *Sal* is ignored.

Repeating the above, but using (49) instead of (48), we get

$$M_V Q' \equiv B_{Emp} \dot{-}^{\bullet} (\hat{\pi}_{Bid,Op,Time,Sal}\hat{B}^{\bullet}_{Emp} \hat{\cup} \hat{\pi}_{Id,Bal,Sex}\hat{\sigma}_{Time<NOW-2yrs\vee (NOW-2yrs<Time\leq NOW-1yrs\wedge(Sex=M\vee Sal<30k))\vee Time>NOW-1yrs}\hat{B}^{\bullet}_{Emp}$$

$$\begin{aligned} M_V Q' \equiv B_{Emp} \dot{-}^{\bullet} (&\hat{\pi}_{Bid,Op,Time,Sal}\hat{B}^{\bullet}_{Emp} \\ &\hat{\cup}\ \hat{\pi}_{Id}\hat{\sigma}_{Time<NOW-7yrs\vee(NOW-6<Time\leq NOW-2yrs\wedge Sex=M)\vee (NOW-2yrs<Time\leq NOW-1yrs\wedge(Sex=M\vee Sal<30k))\vee Time>NOW-1yrs}\hat{B}^{\bullet}_{Emp} \\ &\hat{\cup}\ \hat{\pi}_{Bal}\hat{\sigma}_{Time\leq NOW-5yrs\vee(NOW-5<Time\leq NOW-2yrs\wedge Sex=M)\vee (NOW-2yrs<Time\leq NOW-1yrs\wedge(Sex=M\vee Sal<30k))\vee Time>NOW-1yrs}\hat{B}^{\bullet}_{Emp} \\ &\hat{\cup}\ \hat{\pi}_{Sex}\hat{\sigma}_{Time\leq NOW-6yrs\vee(NOW-6<Time\leq NOW-2yrs\wedge Sex=M)\vee (NOW-2yrs<Time\leq NOW-1yrs\wedge(Sex=M\vee Sal<30k))\vee Time>NOW-1yrs}\hat{B}^{\bullet}_{Emp}) \end{aligned}$$

This example was constructed to illustrate several points that arise when querying a backlog vacuumed according to a projection based specification. Typically, attributes deemed expendable will not be involved as heavily as here. □

Several observations can be made about the relationship between an original query, Q, its modification, $M_V Q$, and its modification using the technique of ignoring conditions that cannot be evaluated due to vacuuming, $M_V Q'$. There are four cases to consider. First, if the selection criteria of Q and the projection on the backlog requested by Q only involve non-vacuumed parts of the backlog, $M_V B_R$, then the technique will not be applied, and $M_V Q' \equiv M_V Q$ so that $M_V Q' \equiv Q$. Second, if selection criteria of Q still do not involve vacuumed parts of the backlog, but attributes requested in the final result of Q have been subjected to vacuuming, then the tuples of the result of $M_V Q$, will be all subtuples of the tuples of the result of Q on the original backlog. Third, in the case where attributes in the selection criteria of Q have been subjected to vacuuming, but no attributes requested in the result of Q have been subjected to vacuuming, the technique applies. Ignoring conjunctive criteria of Q, we have $M_V Q' \succ M_V Q$. Disjunctive criteria that cannot be evaluated are already ignored in $M_V Q$. Fourth, in the final case—where the (conjunctive) criteria of Q and the requested attributes include attributes that have been subjected to vacuuming—a subset of the tuples of the result of $M_V Q'$, possibly with vacuuming-null values, correspond to tuples, without vacuuming-null values, of the result of Q (i.e., only a subset of the tuples of the result of $M_V Q'$ are subtuples of tuples of the result of Q.

8 General Selection/Projection Based Vacuuming

General selection/projection based vacuuming (GSP vacuuming) is vacuuming according to CLASS 4. It unifies selection and projection based vacuuming. The syntax of allowed query expressions is $\pi_A \sigma_F B_R$ where A is a list of attributes, and F is a selection criterion involving any set of attributes of B_R.

GSP vacuuming generalizes projection based vacuuming by not restricting selection criteria to involve only the attribute *Time*. Instead, multi-dimensional selections are possible, and GSP vacuuming can be understood as projection based vacuuming with "multiple time attributes". GSP vacuuming can be used for removal of any set of attribute values from tuples that satisfy some condition that can be specified in terms of the attributes of the backlog (figure 6).

Consider this list:

$$
\begin{array}{ll}
V_2 & \rho(B_{Emp}) : \sigma_{NOW-4yrs<Time\leq NOW-2yrs \wedge Sex=F} B_{Emp} \\
V_{11} & \kappa(B_{Emp}) : \sigma_{Time>NOW-4yrs \wedge Bal\neq 0} B_{Emp} \\
V_{12} & \rho(B_{Emp}) : \pi_{Sex,Sal} \sigma_{Time\leq NOW-2yrs} B_{Emp} \\
V_{13} & \rho(B_{Emp}) : \pi_{Sex,Sal} \sigma_{Time\leq NOW-6yrs \wedge Sal\leq 40k} B_{Emp}
\end{array}
$$

Specification $\{V_2, V_{11}\}$ is selection based; V_{12} is projection based; V_{13} is GSP based and so is $V = \{V_2, V_{11}, V_{12}, V_{13}\}$.

When updating a specification of CLASS 4 for B_R, it is possible for the execution of one part of the specification to rely on data of B_R which have already been vacuumed.

EXAMPLE: Assume that V_{13} has been specified at some earlier point in time and that we now want to augment the specification for B_{Emp} with this specification:

$$
V_{14} \qquad \rho(B_{Emp}) : \pi_{Bal} \sigma_{30k\leq Sal\leq 50k} B_{Emp}
$$

V_{14} removes values of attribute *Bal* if the condition on attribute *Sal* is satisfied. However, vacuuming according to V_{13} can have made it impossible to evaluate the condition: there can exist tuples in $M_{V_{13}} B_{Emp}$ more than six years old and with missing values for attributes *Sex* and *Sal*. For these, we know only that the values of *Sal* are, at most, 40k: we do not know whether they are at least 30k. Thus, we lack information necessary to perform the augmented specification. □

At specification time, a VS is required to perform checks to detect this potential problem of intra-specificational conflicts, and the DBA is required to resolve possible conflicts.

The general procedure for constructing $M_V B_R$ for a GSP based specification is, as can be expected, similar to the one for projection based specifications. To eliminate overlaps, we utilize a generalization of the equivalence (45)

$$
\hat{\pi}_{A_1} \hat{\sigma}_{F_1} \hat{R}^\bullet \mathbin{\hat{-}} \hat{\pi}_{A_2} \hat{\sigma}_{F_2} \hat{R}^\bullet \mathrel{\hat{\equiv}} (\hat{\pi}_{A_1} \hat{\sigma}_{F_1 \wedge \neg F_2} \hat{R}^\bullet \mathbin{\hat{\cup}} \hat{\pi}_{A_1 \setminus A_2} \hat{\sigma}_{F_1 \wedge F_2} \hat{R}^\bullet) \mathbin{\hat{-}} \hat{\pi}_{A_2} \hat{\sigma}_{F_2} \hat{R}^\bullet \tag{50}
$$

where there is no overlap between the keep and delete parts of the right hand side (i.e., either the selection criteria are mutually exclusive or the projections are disjoint).

Again, we have the four forms: SNF$^-$, ANF$^-$, SNF$^+$, and ANF$^+$. We will first derive an SNF$^+$ version and then derive an ANF$^+$ version (SNF$^-$ and ANF$^-$ versions are derived similarly).

Now, there are multiple attributes involved in selection criteria of V, and instead of time intervals, we have multidimensional boxes. The space—where attributes participating in selection criteria of V are the dimensions—is partitioned into maximal, mutually disjoint boxes so that only one single projection of the original backlog expression is valid in each box. Let F_i denote the selection criteria of the i'th box, and let A_i^+ denote the remaining attributes (i.e., attributes of B_R not in A_i^+ have vacuuming null values in box i). The modified backlog expression in SNF$^+$ is given as

$$M_V B_R \equiv \nu\eta\widehat{\bigcup}_{i=1}^{n} \hat{\pi}_{A_i^+}\hat{\sigma}_{F_i}\hat{B}_R^{\bullet} \tag{51}$$

In the case of ANF$^+$, isolate for each attribute the (composite) selection criterion, F_i, that specify vacuuming for the attribute. Collect attributes with equivalent selection criteria, and denote the resulting sets of attributes and corresponding selection criteria A_i and F_i, i = 1,2, ...,m, respectively. The formula then becomes

$$M_V B_R \equiv \nu\eta\widehat{\bigcup}_{i=1}^{m} \hat{\pi}_{A_i}\hat{\sigma}_{\neg F_i}\hat{B}_R^{\bullet} \tag{52}$$

EXAMPLE: Now, let us transform B_{Emp} into the SNF$^+$ version of $M_{V_{13}}B_{Emp}$. The selection criteria of V_{13} involve two attributes; therefore, we have two-dimensional boxes: $F_1' = Time \in [t_{init}; NOW - 6yrs] \wedge Sal \leq 40k$, $A_1' = Id, Bal, Bid, Op, Time$; $F_2' = Time \in [t_{init}; NOW - 6yrs] \wedge Sal > 40k$, $A_2' = Id, Sal, Bal, Bid, Op, Sex, Time$; $F_3' = Time \in [NOW - 6yrs; NOW] \wedge Sal \leq 40k$, $A_3' = Id, Sal, Bal, Bid, Op, Sex, Time$; $F_4' = Time \in [NOW - 6yrs; NOW] \wedge Sal > 40k$, $A_4' = Id, Sal, Bal, Bid, Op, Sex, Time$. Boxes F_2', F_3', and F_4' are not maximal because the same projection is valid for each. Thus, we define $F_1 = F_1'$, $A_1 = A_1'$, $F_2 = F_2' \vee F_3' \vee F_4'$, $A_2 = A_2'$. Consequently,

$$\begin{aligned} M_{V_{13}}B_{Emp} \equiv\ & \nu\eta(\hat{\pi}_{Id,Bal,Bid,Op,Time}\hat{\sigma}_{Time \leq NOW-6yrs \wedge Sal \leq 40k}\hat{B}_{Emp}^{\bullet} \\ & \hat{\cup}\ \hat{\pi}_{Id,Sal,Bal,Bid,Op,Sex,Time}\hat{\sigma}_{Time > NOW-6yrs \vee Sal > 40k}\hat{B}_{Emp}^{\bullet}) \end{aligned}$$

□

EXAMPLE: To illustrate querying of backlogs vacuumed according to a specification in CLASS 4, assume that V_{13} is in effect and consider this query:

$$Q = \pi_{Sex,Id}\sigma_{30k \leq Sal \leq 50k}B_{Emp}$$

Using the expression for $M_{V_{13}}B_{Emp}$ above, we get an expression $M_{V_{13}}Q$ equivalent to Q w.r.t. V_{13}:

$$M_{V_{13}}Q \equiv \nu\eta\hat{\pi}_{Id,Sex}\hat{\sigma}_{(Time > NOW-6yrs \wedge 30k \leq Sal \leq 50k) \vee 40k < Sal \leq 50k}\hat{B}_{Emp}^{\bullet}$$

In box F_1, the condition of Q on attribute Sal fails due to vacuuming, and nothing is retrieved from this box. In box F_2, the condition of Q on Sal and that of the box are simply combined. By leaving out the condition on Sal in box F_1, we get a more informative expression:

$$M_{V_{13}}Q' \equiv \nu\eta(\hat{\pi}_{Id}\hat{\sigma}_{Time \leq NOW-6yrs \wedge Sal \leq 40k}\hat{B}_{Emp}^{\bullet}\ \hat{\cup}\ \hat{\pi}_{Id,Sex}\hat{\sigma}_{(Time > NOW-6yrs \wedge 30k \leq Sal \leq 50k) \vee 40k < Sal \leq 50k}\hat{B}_{Emp}^{\bullet})$$

In box F_1, the condition of Q on attribute *Sal* cannot be evaluated, and attribute *Sex* has been vacuumed. In box F_2, the condition of Q on *Sal* and that of the box are simply combined. □

The observations made in the previous section about the possible relationships between queries and modified queries on backlogs vacuumed according to specifications in CLASS 3 are also valid when backlogs are vacummed according to specifications in CLASS 4.

9 Time Slice Based Vacuuming

In this section, we discuss specifications of CLASS 5 and CLASS 6. The syntax is $B_R(t) - B_R(t) \ltimes \pi_{Time}R(t)$ and $\pi_A(B_R(t) - B_R(t) \ltimes \pi_{Time}R(t))$, respectively. These classes were created to provide the simplest possible vacuuming facilities suitable for backlogs where time slice queries are important. However, note that κ specifications of the format of CLASS 5 and CLASS 6 are awkward and of limited use. Therefore, we will not consider κ specifications in this section.

CLASS 5 is a subclass of CLASS A, and the idea is to remove an old part of a backlog, but retain any tuples that are needed to compute a time slice newer than a given time. Consider this specification:

$$V_{15} \qquad \rho(B_{Emp}) : B_{Emp}(NOW - 2yrs) - B_{Emp} \ltimes \pi_{Time}Emp(NOW - 2yrs)$$

Here, all tuples of B_{Emp} more than two years old are removed with the exception that if a tuple is used to construct $Emp(NOW - 2yrs)$, it is retained. This ensures that any time slice query after $NOW - 2yrs$ is unaffected by vacuuming.

CLASS 6 generalizes CLASS 5 by allowing projections of tuples instead of whole tuples to be removed, and it is a subclass of CLASS C. A sample CLASS 6 specification would be

$$V_{16} \qquad \rho(B_{Emp}) : \pi_{Id}(B_{Emp}(NOW - 8yrs) - B_{Emp} \ltimes \pi_{Time}Emp(NOW - 8yrs))$$

$$V_{17} \qquad \rho(B_{Emp}) : \pi_{Bal}(B_{Emp}(NOW - 6yrs) - B_{Emp} \ltimes \pi_{Time}Emp(NOW - 6yrs))$$

$$V_{18} \qquad \rho(B_{Emp}) : \pi_{Sal}(B_{Emp}(NOW - 4yrs) - B_{Emp} \ltimes \pi_{Time}Emp(NOW - 4yrs))$$

$$V_{19} \qquad \rho(B_{Emp}) : \pi_{Sex}(B_{Emp}(NOW - 2yrs) - B_{Emp} \ltimes \pi_{Time}Emp(NOW - 2yrs))$$

Consider another specification:

$$V_{20} \qquad \rho(B_{Emp}) : \pi_{Sal}\sigma_{Time \leq NOW-4yrs}B_{Emp}$$

$$V_{21} \qquad \kappa(B_{Emp}) : Emp(NOW - 4yrs)$$

V_{20} removes values of attribute *Sex* from tuples more than two years old, and V_{21} disallows removal of tuples (and their values) needed to compute $Emp(NOW - 4yrs)$. Thus, at first sight $V = \{V_{20}, V_{21}\}$ seems totally equivalent to V_{18}. However, a κ-specification has a global effect, so once we add, for example, V_{19} to both V and V_{18}, the equivalence disappears.

So far, we have only said that the symbol t in the syntax $B_R(t) - B_R(t) \ltimes \pi_{Time} R(t)$ for the two classes denotes expressions that evaluate to elements of the domain *TTime* (i.e., *Time* values). Let us analyze which restrictions on the possible orderings should be enforced. To do so, assume the syntax is $B_R(x) - B_R(y) \ltimes \pi_{Time} R(z)$ and that x, y, and z are evaluated expressions. There are 13 different orderings of x, y, z. We want the specification to vacuum all data with *Time* value less than or equal to x, and we want time slice queries after *Time* z to be unaffected by vacuuming. To achieve this, we require that $x \leq z$. If $y < x$, time slice becomes unsafe because needed tuples with *Time* values in $]y; x]$ (if any!) are not retained, but are "mistakenly" vacuumed. Thus, we also require that $x \leq y$. We are then left with the valid orderings: $x < y < z$, $x < z < y$, $x = y = z$, $x = y < z$, $x = z < y$, $x < y = z$. Note that y can be changed to x in a specifications with $x < y$ without changing the meaning of the specification. It can also be observed that the advantages from having $x = z$ instead of $x < z$ (i.e., more unaffected time slices) seem to outweight the disadvantages (i.e., less data vacuumed)—removal of more data is better achieved by increasing x. Consequently, we have $x = y = z$ in typical usage.

The problem of enforcing growing specifications of CLASS 5 and CLASS 6 is nonexistent because, by definition, any specification of these classes is growing[7].

The formula for computing a modified backlog $M_V B_R$ from a one element specification V of CLASS 5 with $x = y = z$ is simple. From the general formula (2), we have

$$M_V B_R \equiv B_R - (B_R(x) - B_R(x) \ltimes \pi_{Time} R(x)) \tag{53}$$

$$M_V B_R \equiv (B_R(x) \cup \sigma_{Time > x} B_R) - (B_R(x) - B_R(x) \ltimes \pi_{Time} R(x))$$

$$M_V B_R \equiv \sigma_{Time > x} B_R \cup B_R(x) \ltimes \pi_{Time} R(x) \tag{54}$$

Formula (53) tells what to remove, and formula (54) tells what to retain.

For an arbitrary specification (still with $x = y = z$), the formulas are similar: instead of using x, we use the value α defined as $\max(\{x_i\}_{i=1}^n)$ where x_i, $i = 1, 2, \ldots, n$ are time values of ρ specifications. This is so because

$$t_1 \geq t_2 \Rightarrow B_R(t_1) - B_R(t_1) \ltimes \pi_{Time} R(t_1) \succ B_R(t_2) - B_R(t_2) \ltimes \pi_{Time} R(t_2) \tag{55}$$

Specifications of CLASS 6 can be perceived as a generalization of specifications of CLASS 5. Dealing with CLASS 6, we can simply treat each attribute of R as we treated a tuple dealing with CLASS 5. Each attribute can be treated in isolation, and the results from doing so are collected by the "union" operator ($\hat{\cup}$). The i'th ($1 \leq i \leq n$) element of a specification has a projection list of the form $A_{i_1}, A_{i_2}, \ldots, A_{i_{k_i}}$. The i'th element is split into i_{k_i} elements, $\pi_{A_j}(B_R(x) - B_R(x) \ltimes \pi_{Time} R(x))$, $j = i_1, i_2, \ldots, i_{k_i}$. This process gives us $\Sigma_{i=1}^n k_i$ elements. They are collected into m groups where m is the number of attributes of B_R. The l'th of these groups has elements of the form $\pi_{A_l}(B_R(x_l) - B_R(x_l) \ltimes \pi_{Time} R(x_l))$. A time value, α_l, is selected as above. Elements with $\alpha_l = t_{init}$ are discarded, and elements with identical α_l are merged. This leaves k sets of attributes, A_l^-, and corresponding time values, α_l, $l = 1, 2, \ldots, k$. Now, the ANF$^-$ version of $M_V B_R$ is given by

$$M_V B_R \equiv \nu \widehat{\bigcup}_{l=1}^{k} \hat{\pi}_{A_l^-}(\hat{B}_R^{\bullet}(\alpha_l) \hat{-} \hat{B}_R^{\bullet}(\alpha_l) \hat{\ltimes} \hat{\pi}_{Time} \hat{R}^{\bullet}(\alpha_l)) \tag{56}$$

[7] We assume that all *TTime*-valued expressions are non-decreasing functions of time.

The ANF$^+$ version for CLASS 6 is obtained by applying to each term the transformation that lead from formula (53) to formula (54) and by including the attributes, A^d, which were discarded above. Doing so, we get

$$M_V B_R \equiv \nu\eta\left(\widehat{\bigcup}_{l=1}^{k} \hat{\pi}_{A_l^-}(\sigma_{Time>\alpha_l}\hat{B}_R^\bullet(\alpha_l)\hat{\cup}\hat{B}_R^\bullet(\alpha_l)\hat{\ltimes}\hat{\pi}_{Time}\hat{R}^\bullet(\alpha_l))\ \hat{\cup}\ \hat{\pi}_{A^d}\hat{B}_{Emp}^\bullet\right) \tag{57}$$

Due to the format of specifications of CLASS 5 and CLASS 6, the SNF$^-$ and SNF$^+$ versions of $M_V B_R$ become more complicated than their ANF counterparts, and they are thus not competitive (see the example below).

Assume that $V = \{V_{16}, V_{17}, V_{18}, V_{19}\}$ is in effect. Then, the modified backlog can be expressed as (ANF$^-$ version)

$$\begin{aligned} M_V B_{Emp} \equiv\ & \nu(\hat{\pi}_{Id}(\hat{B}_{Emp}^\bullet(NOW-8yrs) \hat{-} \hat{B}_{Emp}^\bullet \hat{\ltimes} \hat{\pi}_{Time}\widehat{Emp}^\bullet(NOW-8yrs)) \\ & \hat{\cup}\ \hat{\pi}_{Bal}(\hat{B}_{Emp}^\bullet(NOW-6yrs) \hat{-} \hat{B}_{Emp}^\bullet \hat{\ltimes} \hat{\pi}_{Time}\widehat{Emp}^\bullet(NOW-6yrs)) \\ & \hat{\cup}\ \hat{\pi}_{Sal}(\hat{B}_{Emp}^\bullet(NOW-4yrs) \hat{-} \hat{B}_{Emp}^\bullet \hat{\ltimes} \hat{\pi}_{Time}\widehat{Emp}^\bullet(NOW-4yrs)) \\ & \hat{\cup}\ \hat{\pi}_{Sex}(\hat{B}_{Emp}^\bullet(NOW-2yrs) \hat{-} \hat{B}_{Emp}^\bullet \hat{\ltimes} \hat{\pi}_{Time}\widehat{Emp}^\bullet(NOW-2yrs))) \end{aligned}$$

Note the direct correspondence between the structure of this formula and that of its specification. In the interval between NOW and $NOW - 2yrs$, any query (including any time slice) can be answered. For other intervals, this is not true. For example, between $NOW - 4yrs$ and $NOW - 6yrs$, all data of attributes *Id* and *Bal* are present, but occurrences of values of *Sal* are only present if they are part of $Emp(NOW - 4yrs)$, and occurrences of values of *Sex* are only present if they are part of $Emp(NOW - 2yrs)$.

In general, a modified backlog can be partitioned into intervals (with the time values α_l of formula (56) as boundaries) such that a subset, S, of the terms of the backlog expression (ANF$^-$ version) contribute to the vacuuming for that interval. The terms for an old interval will contain the terms for a more recent interval. With V in effect, the intervals are $[t_{init}; NOW-8yrs]$, $]NOW-8yrs; NOW-6yrs]$, $]NOW-6yrs; NOW-4yrs]$, $]NOW-4yrs; NOW-2yrs]$, and $]NOW-2yrs; NOW]$. In interval $]NOW-4yrs; NOW-2yrs]$, all values of attribute *Sex* have been removed except for the ones in tuples among $B_{Emp}(NOW-2yrs) \ltimes \pi_{Time}Emp(NOW-2yrs)$. Now, let a query, Q be given as $\sigma_{Sex=F}Emp(NOW-3yrs)$. Then, using this observation, we can express the modified query as

$$M_V Q \equiv \sigma_{Sex=F}(B_{Emp}(NOW-2yrs) \ltimes \pi_{Time}Emp(NOW-2yrs))(NOW-3yrs)$$

The modified query retrieves all tuples with $Sex = F$ that were part of *Emp* two years ago, and that were not modified between two and three years ago (i.e., they were also part of *Emp* three years ago). Generally, when Q is a time slice query, the interval that the time slice belongs to is identified together with the vacuuming specifications relevant to that interval, and the backlog modified with these is time sliced.

10 General Vacuuming

In this section, we discuss CLASS 7 and CLASS 8. For the former, the syntax is $Q(B_R)$ where Q is any query expression involving B_R, including time slices of B_R. For the latter, the syntax is $Q(B_{R_1}, B_{R_2}, \ldots, B_{R_n})$ where

$R_1, R_2, \ldots, R_n$ are all user-defined relations, and Q is an arbitrary query restricted to the standard operators of DM/T.

In this section, we mainly exemplify the possibilities of the two classes of specifications, and we do not derive specific formulas for $M_V B_R$—instead, we refer to the coverage of the general case in section 5.

Examples of specifications of CLASS 7 (and not in previous classes, of course) can be obtained by composing specifications of CLASS I and CLASS II; thus, previous examples apply. To exemplify further, assume that a decision is made to retain only values of attributes *Id*, *Time*, and *Op* of modification requests more than two years old for employees with a nonnegative balance throughout the last year. The expression $\pi_{Id}\sigma_{Time \geq NOW-1yrs} B_{Emp} - \pi_{Id}\sigma_{Time \geq NOW-1yrs \wedge Bal<0} B_{Emp}$ retrieves the values of attribute *Id* for all the qualifying employees, and the part of B_{Emp}, where tuples belonging to qualifying employees must be removed, is retrieved by the expression $\pi_{Id,Time,Op}\sigma_{Time<NOW-2yrs \wedge Op=M} B_{Emp}$. Using the semi-join, we obtain

$$V_{22} \qquad \rho(B_{Emp}) : \pi_{Id,Time,Op}\sigma_{Time<NOW-2yrs \wedge Op=M} B_{Emp} \ltimes \pi_{Id}\sigma_{Time \geq NOW-1yrs} B_{Emp} - \pi_{Id}\sigma_{Time \geq NOW-1yrs \wedge Bal<0} B_{Emp}$$

To illustrate specifications of CLASS 8, we expand our sample database, $\mathcal{DB}$, with a department relation, *Dep*, with schema:

$$Dep(Id : Int; Job : \{D, R, A\}; Off : \{N, C, W, L\})$$

where *Job* can be either Development (D), Research (R), or Administration (A), and *Off* can be "No office" (N), "Core office" (C), "Window office" (W), and "Large window office" (L).

For CLASS 1 - CLASS 7 vacuuming specifications have been based solely on the values of the attributes of the relations to vacuum. CLASS 8 adds the capability of basing the vacuuming of one relation on the attribute values of other relations. In analogy with derived fragmentation of global relations into local fragments in distributed databases, this can be called derived vacuuming.

To illustrate, the following specifies removal of all tuples in *Emp* if they represent an employee currently without an office:

$$V_{23} \qquad \rho(B_{Emp}) : B_{Emp} \ltimes \pi_{Id}\sigma_{Off=N} Dep$$

As another example, suppose we only want to be able to ask time slice based queries within the last two years on employees in development and research, but that we still want full capabilities for employees that have only been in administration. Then, we can issue this specification:

$$V_{24} \qquad \rho(B_{Emp}) : B_{Emp}(NOW - 2yrs) \ltimes \pi_{Id}\sigma_{Job=D,R} B_{Dep}(NOW - 2yrs) - Emp(NOW - 2yrs)$$

CLASS 8 is a very general class of specifications. Its extra power—the possibility of using semi-joins, joins, and Cartesian products involving other backlogs—has the complication that both intra-specificational and inter-specificational conflicts can occur.

As another complication, the expression of a modified backlog and thus of modified queries can become very complex. If this occur, we find it likely that less ambitious vacuuming, that is easier to express, will be preferable.

In conlusion, if derived vacuuming is needed, the potential complexity can be dealt with in two combinable ways. First, the vacuuming subsystem designer may allow only a limited subset of CLASS 8—numerous subclasses beyond the scope of this presentation are possible. Second, the DBA can exercise a strict discipline and avoid vacuuming specifications that result in complex modified backlog expressions.

11 Conclusion

Temporal databases supporting transaction time are ever-growing. In this paper, we have considered how to cope with the constant growth by vacuuming data without changing history. We have presented a framework for vacuuming subsystems for temporal databases that adhere to this principle. The framework is a useful tool for designers of vacuuming subsystems of future DBMS's supporting transaction time. It provides an understanding of the issues involved in the design of a specific vacuuming subsystem, it outlines a wide range of choices of functionality of a specific vacuuming subsystem along with the consequences of the particular choices, and it gives a foundation for correct and cooperative query processing.

In addition to the physical removal of data, a vacuuming subsystem should augment the functionality of a temporal DBMS in two ways. First, it should support the specification of vacuuming. Second, it should allow for correct and cooperative processing of queries against base relations that have been vacuumed.

We proposed two types of specifications, namely removal (ρ) and keep (κ) specifications, and we presented three classifications of vacuuming specifications. The first and second classifications were both used for characterization of the 8 classes of the third classification. The first classification made use of two dimensions to distinguish between specifications. Thus, specifications that were based on the *Time* attribute alone were classified differently from those based on an arbitrary set of attributes. In addition, specifications that removed complete tuples and specifications that removed individual attribute values were classified differently. In the second classification, we distinguished between backlog based and time slice based specifications. We identified correctness criteria for specifications: they must be growing and conflicts with other specifications, integrity constraints, view definitions, and application programs must be resolved.

A foundation for correct and cooperative query processing was established by providing a set of rules for modifying queries that access vacuumed parts of base relations into similar queries that can be answered correctly because they avoid access to vacuumed data. We treated each of the 8 classes in turn. For each of the classes with tuples as the unit of deletion, we developed relational algebra formulas to express vacuuming of backlog relations. For the remaining classes with attribute value occurrences as the unit of deletion, we extended the relational algebra to a tagged relational algebra and developed formulas to express vacuuming.

12 Future Research

As pointed out previously, we believe this to be the first paper to consider how to delete data from temporal databases without changing history. Consequently, we find it appropriate to point to several important areas of future research. We devote subsections (1) to extension of the framework to include off-line archival as an intermediate between on-line storage and irreversible deletion, (2) to the incorporation of more powerful vacuuming capabilities based on a new type of relation, (3) to issues related to how the cooperativeness of a VS can be further improved, and (4) to the integration of vacuuming into IM/T, the implementation model for DM/T.

12.1 Extending with Off-Line Archival

In conventional DBMS's, complete transaction logs are sometimes stored on magnetic tape and never deleted. In addition, temporal databases that have already been vacuumed may still not fit in on-line storage. These observations suggest that off-line archival should be included into the framework for vacuuming. Here, we will mention some of the issues that arise when we include both ρ, κ, and τ specifications, the last one identifying data that are to be moved from expensive and fast on-line storage, typically magnetic disc, to cheap and slow off-line mass storage, typically magnetic tape.

Theoretically, the semantics of τ specifications differ from those of ρ specifications. While removal of data (ρ) is inherently irreversible, moving data from on-line to off-line storage (τ) is reversible. This has two implications: first, the concept of growing specification does not apply to τ specifications the same way it applied to specifications with ρ and κ—growth was required exactly because of the irreversibility of deletions; second, an inverse operator, τ^{-1}, must be offered to reverse the migration from on-line to off-line storage. This operator may be employed by the system during several tasks: when dealing with *moving* τ specifications, when answering queries that need access to off-line data, and when specifications are updated. In the last case, when data previously archived by a τ specification, $\tau(B_R) : Q$, are wanted on-line, the DBA removes specification $\tau(B_R) : Q$, and triggers the system to issue the operation $\tau^{-1}(Q)$ that brings the desired data on-line.

In practice, the semantics appropriate for τ specifications may depend on the particular type of off-line media, the level of operator service that is provided, and the nature of the applications running against the DBMS. For example, if it will take hours or days to gain access to data off-line and if applications require response times in the range of seconds, then it makes sense to require that τ specifications be growing.

When introducing a new construct, its interaction with and effect on existing constructs must be considered in detail. To illustrate the issues, note that data items can be *specified* or *referenced*. For example, a semi-join, $R \ltimes S$, specifies data part of the left hand side argument, R, and references data part of the right hand side argument, S, in order to serve as a specification. Several questions arise due to the possibilities of overlaps between the data accessed (specified/referenced) by the various constructs.

- Can the same data be both specified by some τ specification while being referenced by another specification? If so, this would mean that it may be necessary to move data from off-line to on-line storage in order to evaluate

a τ specification.

- Can a τ and a κ specification specify the same data? If so, should the common data be stored off-line, on-line, or should it be left unspecified? That is, is the semantics of κ that data specified must be stored on-line or merely that data specified must not be deleted?

- Can the same data be specified by a τ specification and referenced by a κ specification at the same time? If so, moving data to on-line storage may be necessary to evaluate the κ specification. The same question arises when we replace κ specifications with ρ specifications, view definitions, and query expressions of application programs.

- Can the same data be specified by a τ and a ρ specification at the same time? If so, what should the semantics be? The ρ specification can override the κ specification, but the τ specification generally cannot override the ρ specification. For example, this is so if the ρ specification were in effect before the τ specification became effective. Similar questions arise when view definitions and query expressions of application programs are substituted for ρ specifications.

When it has been resolved which specifications are valid and what their semantics are, appropriate responses to queries that access data stored off-line must be considered. Prior to the introduction of τ specifications, a query, Q, was modified so that it did not access data removed by ρ specifications, and one or several such modifications were presented to the user. First, it is now possible that a query, Q, must be modified into $M_\rho Q$ due to ρ specifications; second, due to τ specifications, it is possible that $M_\rho Q$ cannot be answered without the access to archived data. Because this can result in an unacceptable delay, there is also a need to present at least one modification of $M_\rho Q$, $M_{\rho\tau} Q$, that avoids the access to archived data. Consequently, when τ specifications are allowed, two distinguished types of modified queries will be presented to the user. A user will be able to issue a query of the first type, get an immediate answer, and can then choose to issue a query of the other type and engage in other tasks while waiting for the result.

12.2 Statistics Views

The kind of vacuuming we have seen so far results in data being deleted from existing relations. For some purposes, more advanced capabilities are desirable. Assume for example that, due to space constraints, only the last four years of *Emp* can be retained (V_1), but that it is important to know, at any time, the number of employees in the company database ever since its creation. This ***cannot*** be accomplished by the previously introduced mechanisms. However, by introducing the concept of *S-view* (Statistics view), this becomes possible. In the case of this example, we could issue this definition (see the appendix for a definition of ξ):

`define S-view:` $EmpSum$ `as:` $\pi_{Sum} \xi_{-,Sum=count(Id)} B_{Emp}$

In response to this definition, the DBMS creates and incrementally [15] maintains a backlog, B_{EmpSum}, with this schema:

$$B_{EmpSum}(Op : \{I, D, M\}; Bid : SURROGATE; Time : TTIME; Sum : INT)$$

To see how an S-view differs from base relation, view, and snapshot [1], first note that the part of S-view B_{EmpSum} more than four years old cannot be derived from B_{Emp} due to vacuuming but must be computed and stored before vacuuming takes place. Thus, unlike a view, an S-view is not derivable from base data (e.g., B_{EmpSum} cannot be derived from $M_{V_1}B_{Emp}$). Unlike a snapshot, an S-view is not independent of the base relations, from which it is derived. Updates to the base relations must be reflected in the S-view. Finally, unlike a base relation, an S-view is derived from another relation (e.g. when B_{Emp} is updated, B_{EmpSum} must be updated at most four years later).

S-views can be seen as a natural extension of the kind of statistics that reflect parts of backlogs before they were vacuumed and still fits the schemas of the backlogs. Using the aggregation mechanisms of the query language, an S-view for a base relation can maintain summaries that, due to vacuuming, are no longer obtainable from the base relation; and using advanced operators (e.g., the Σ operator [13]), the DBA is able to vacuum bulks of detailed data and just retain extracts of interest. Extracts can indicate trends and patterns of deleted data, can be exceptional and deviating data, or can be both.

12.3 Increased Cooperativeness

In this paper, we provided the foundation for cooperative query processing in temporal databases supporting transaction time where relations can be vacuumed. Within CLASS B, the query modifier returned a set of one or more equivalent modified queries, and outside CLASS B it, in addition, applied the technique of ignoring selection criteria that could not be evaluated due to vacuuming.

It is an interesting topic how to extend the query modifier's capabilities of returning understandable and useful queries when the original query cannot be evaluated due to vacuuming.

We use the example in section 2 to illustrate this idea. Figure 1 illustrates two equivalent modifications of query $Q = \sigma_{Sal>30k}B_{Emp}$ with vacuuming specification $V = \{V_1, V_2, V_3\}$ in effect. We can simplify the two modifications by strengthening the restrictions implied by V, by weakening the restriction of Q, or both. For example, if we assume that sparse deletion requests are of little interest, then we may delete the terms `(Time <= NOW - 4yrs and Op = D and Sal >= 30k)` and `Op = D` from the second alternative of figure 1 and achieve the shorter, more specialized, and perhaps more useful query:

```
select [(NOW - 4yrs < Time <= NOW - 2yrs and Sex = M and Sal >= 30k) or
   (NOW - 2yrs < Time and Sal >= 30k)] B[Emp]
```

Similarly, we can delete the term `Op = D` from the first alternative and get this query:

```
select[Sal >= 30k and Time > NOW - 4yrs and (Time > NOW - 2yrs or Sex = M)] B[Emp]
```

Techniques for query generalization/specialization [19, 6], which have been developed for other purposes, can prove applicable as the starting point towards improved cooperativeness. However, additional issues must be addressed.

For example, it is not obvious how to determine which generalizations/specializations out of the vast number of possibilities are useful (i.e., easy to understand and of interest to the user). In addition, it can be necessary to be capable of choosing a few queries from a larger set of queries that seem useful.

In conclusion, though improved cooperativeness by generalization/specialization is important, it is far from trivial.

12.4 Implementation Issues

We have been concerned with semantic and conceptual issues of vacuuming, and actual physical vacuuming has been beyond the scope of the presentation. Integration of vacuuming into the implementation model, IM/T [16], of DM/T is an interesting area of current research. IM/T offers differential computation of views based on results of previous computations stored as pointer structures or actual data in an indexed cache. IM/T also offers cache update strategies ranging from eager, via threshold triggered, to lazy update. Below, we list 5 open problems.

1. How can the capabilities of IM/T best be exploited to allow for differential physical vacuuming? Vacuuming specifications can be perceived as views, and as such, they can be computed and stored as pointer structures. Upon doing so, following the pointers will allow for identification of all base records that need to be vacuumed. When vacuuming is done this way, the cached views corresponding to vacuuming specifications have empty extensions, and consequently, it is interesting to explore how differential update of views with empty outset can be performed most efficiently.

2. Which conditions should trigger vacuuming? Vacuuming logically has eager semantics, but physical vacuuming can be done using deferred strategies. Intuitively, it is clear that vacuuming should be done often enough that space shortage is avoided and so that it does not compromise the responsiveness of the system. However, it is a non-trivial issue of research to actually implement the abstract idea.

3. What should be the unit of vacuuming triggered by a condition? There are many possibilities. For example, it is possible to vacuum the whole database, a set of backlogs, a single backlog, and a sub-specification for a backlog.

4. How is the property (when applicable), $M_V Q \equiv_V Q$, best exploited during query processing? In addition, note that there will be intermediate queries (between $M_V Q$ and Q) that also can be processed instead of $M_V Q$.

5. Which reorganizations of physical structures are appropriate during massive (and partly predictable) deletions? Given that the set of vacuuming specifications is relatively stable and given that the rate of update of relations is also stable, it is possible to predict the removals by a VS. It is an interesting issue how to organize physical storage in advance so as to take advantage of the knowledge of deletions.

References

[1] Michel Adiba. Derived relations: A unifyed mechanism for views, snapshots and distributed data. *7th VLDB Conference*, pages 293–305, 1981.

[2] Rakesh Agrawal. Alpha: An extension of relational algebra to express a class of recursive queries. *3rd Data Engineering Conference*, pages 1–11, February 1987.

[3] A. V. Aho, Y. Sagiv, and J. D. Ullman. Efficient optimization of a class of relational expressions. *ACM TODS*, 4(4):435–454, December 1979.

[4] A. V. Aho, Y. Sagiv, and J. D. Ullman. Equivalences among relational expressions. *SIAM Journal of Computing*, 8(2):218–246, May 1979.

[5] R. F. Boyce, D. D. Chamberlin, W. F. King, and M. M. Hammer. Specifying queries as relational expressions: The square data sublanguage. *Comm. ACM*, 18(11):621–628, November 1975.

[6] Surajit Chaudhuri. Generalization as a framework for query modification. *6th Data Engineering Conference*, pages 138–145, February 1990.

[7] E. F. Codd. A relational model of data for large shared data banks. *Comm. ACM*, 13(6):377–387, June 1970.

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

[9] C. J. Date. *An Introduction to Database Systems — Volume II.* The Systems Programming Series. Addison Wesley Publishing Company, first, corrected edition, July 1985.

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

[11] Umeshwar Dayal and Philip A. Bernstein. On the updatability of relational views. *4th VLDB Conference*, pages 368–377, 1978.

[12] A. L. Furtado, K. C. Sevcik, and C. S. Dos Santos. Permitting updates through views of data bases. *Information Systems*, 4:269–283, 1989.

[13] Christian S. Jensen and Leo Mark. Queries on change in an extended relational model. Technical report CS-TR-2299, UMIACS-TR-89-80, Department of Computer Science. University of Maryland, College Park, MD 20742, August 1989. To appear in IEEE Transactions on Knowledge and Data Engineering.

[14] Christian S. Jensen, Leo Mark, and Nick Roussopoulos. Incremental implementation model for relational databases with transaction time. Technical report CS-TR-2275, UMIACS-TR-8963, Department of Computer Science. University of Maryland, College Park, MD 20742, June 1989. To appear in IEEE Transactions on Knowledge and Data Engineering.

[15] Christian S. Jensen, Leo Mark, Nick Roussopoulos, and Timos Sellis. A framework for efficient query processing using caching, cache indexing, and differential techniques in the relational model extended with transaction time. Technical report R-89-45, The University of Aalborg, Institute for Electronic Systems. Dept. of Math. and Comp. Sci., Frederik Bajersvej 7, DK-9220 Aalborg Øst, Denmark, December 1989. Extended version of [16].

[16] Christian S. Jensen, Leo Mark, Nick Roussopoulos, and Timos Sellis. Using caching, cache indexing, and differential techniques to efficiently support transaction time. Technical report CS-TR-2413, UMIACS-TR-90-25, Department of Computer Science. University of Maryland, College Park, MD 20742, February 1990. Submitted for publication.

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

[18] Edwin McKenzie and Richard Snodgrass. Schema evolution and the relational algebra. *Information Systems*, 5(1), April 1990.

[19] Amihai Motro. Query generalization: A technique for handling query failure. *1st International Workshop on Expert Database Systems*, pages 314–325, October 1984.

[20] Jooseok Park and Arie Segev. Using common subexpressions to optimize multiple queries. *4th Data Engineering Conference*, pages 311–319, February 1988.

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

[22] Lawrence A. Rowe and Michael R. (eds.) Stonebraker. The postgres papers. Memorandum UCB/ERL M86/85, University of California, Electronics Research Laboratory, College of Engineering, University of California, Berkeley, CA 94720, June 1987.

[23] Ben Shneiderman and Glenn Thomas. An architecture for automatic relational database system conversion. *ACM TODS*, 7(2):235–257, June 1982.

[24] John Miles Smith and Philip Yen-Tang Chang. Optimizing the performance of a relational algebra interface. *Comm. ACM*, 18(10):569–579, October 1975.

[25] Richard Snodgrass and Ilsoo Ahn. A taxonomy of time in databases. *ACM SIGMOD '85*, pages 236–246, 1985.

[26] Michael Stonebraker. Implementation of integrity constraints and views by query modification. Memorandum ERL-M514, Electronics Research Laboratory, College of Engineering, University of California, Berkeley 94720, March 1975.

[27] Michael Stonebraker. The design of the postgres storage system. *13th VLDB Conference*, pages 289–300, 1987.

[28] Michael Stonebraker and Lawrence A. Rowe. The design of postgres. Memorandum UCB/ERL 85/95, Electronics Research Laboratory, College of Engineering, University of California, Berkeley 94720, California, November 1985.

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

[30] Jeffrey D. Ullman. *Database and Knowledge - Base Systems*, volume II: The New Technologies of *Principles of Computer Science.* Computer Science Press, 1803 Research Boulevard, Rockville, MD 20850, 1988.

A.1 Appendix

To make the paper self-contained, we give a brief description of the extended DM/T query language. The first subsection describes DM/T, and the second describes the aggregate formation operator, ξ.

A.1.1 Transaction Time in the Relational Model, DM/T

DM/T is a transaction time extension of the standard relational model [7, 8]. The model was introduced in [14]; in [13], it was extended to support queries on change; in [15] and [16] efficient implementation is addressed.

The properties of the the time concept offered by DM/T are outlined in figure 9 and are discussed below.

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

Figure 9: Characterization of the time concept offered by DM/T.

Two orthogonal time dimensions have been studied in temporal databases [25]. Logical time models time in the part of reality modeled by a database. Transaction time models time in the part of the reality that surrounds the database, the input subsystem. While logical time is application dependent, transaction time depends only on the DBMS and is inherently application independent.

First, DM/T supports transaction time as opposed to logical time. Second, a domain is regular if the distances between consecutive values of the active domain are identical. Otherwise, the domain is irregular. DM/T supports an irregular time domain. Third, a time domain can be discrete or stepwise continuous. Facts with discrete time-stamps are only valid at the exact times of their time stamps. In contrast, in a stepwise continuous domain facts have an interval of validity. The DM/T time domain has this property (also termed *stability*), because the values of a relation remain the same until the relation is changed by a new transaction. Fourth, DM/T supports true time as opposed to arbitrary time. True time reflects the actual time of the input subsystem while an arbitrary time domain needs only to have a metric and a total order defined on it; the natural numbers is a possible arbitrary time domain. Fifth, DM/T has automatic time-stamping, which is the natural choice for transaction time. Manual, user

supplied time-stamp values are natural for logical time. We have chosen tuple stamping as opposed to attribute value stamping. The major motivation has been to provide a 1.NF model which is a simple and yet powerful extension of the basic relational model.

In order to record detailed temporal data and still be able to use the operators of the basic relational model, we have introduced the concept of a backlog relation. A *backlog*, B_R, for a relation, R, is a relation that contains the complete history of change requests to relation R [21]. Backlog B_R contains three attributes in addition to those of R. Attribute *Id* is defined over a domain of logical, system generated unique identifiers (i.e., surrogates). The values of *Id* represent the individual tuples, termed *change requests*. The attribute *Op* is defined over the enumerated domain of operation types, and values of *Op* indicate whether an insertion (*Ins*), a deletion (*Del*), or a modification (*Mod*) is requested[8]. Finally, the attribute *Time* is defined over the domain of transaction time stamps, *TTIME*, as discussed previously. DM/T automatically generates and maintains a backlog for each base relation (i.e., user defined relations and schema relations). Figure 10 shows the effect on backlogs resulting from operation requests on their corresponding relations. As a consequence of the introduction of time stamps, a base relation is now a function

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

Figure 10: System controlled insertions into a backlog. The function "tuple" returns the tuple identified by its argument.

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 every relation is empty. If R is parameterized with an expression that evaluates to a time value, then the result is the state of R as it was at that point in time. It has no meaning to use a time before the database was initialized and after the present time. If R is used without any parameters, then this indicates the current R. Time sliced relations have an *implicit* time stamp attribute, not shown unless explicitly projected. Note that these features help provide transparency to the naive user. We also introduce the special variable NOW which assumes the time when the query is executed.

[8] On a lower level, modifications are modeled by a deletion followed by an insertion, each with the same time stamp.

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

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

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

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

We adopt a set of precedence rules to simplify the appearance of query expressions. Time slice has the highest precedence and is followed by the other unary operators, all with the same precedence. These, in turn, are followed by binary operators all of which again have the same precedence. Parentheses are used to control precedence in the standard way, and evaluation is from left to right.

A.1.2 The Aggregate Formation Operator

The *aggregate formation* operator, ξ, is used to apply aggregate functions to groups of attribute values. The operator we present here is a variation of the one described in [17] and [2]. 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 resulting value is associated with each tuple in the group as a value of the *att_name* attribute.

The following query generates a relation telling what the average salary is for each department (assuming a *Dep* attribute):

$$\xi_{Dep,Avg\_sal=avg_{Sal}}\ Emp$$

where *Avg_sal* is the new attribute to be generated. One group is generated for each distinct value of *Dept*. X can be any sequence of distinct attributes in relation R.