Vacuuming Temporal Databases

Janne Skyt, Christian S. Jensen

September 4, 1998

TR-32

A TIMECENTER Technical Report

Title	Vacuuming Temporal Databases
Author(s)	Janne Skyt, Christian S. Jensen
Publication History	September 1998. A TIMECENTER Technical Report.

TIMECENTER Participants

Aalborg University, Denmark
Christian S. Jensen (codirector), Michael H. Böhlen, Renato Busatto, Curtis E. Dyreson, Heidi Gregersen, Dieter Pfoser, Simonas Šaltenis, Janne Skyt, Giedrius Slivinskas, Kristian Torp

University of Arizona, USA
Richard T. Snodgrass (codirector), Sudha Ram

Individual participants
Anindya Datta, Georgia Institute of Technology, USA
Kwang W. Nam, Chungbuk National University, Korea
Mario A. Nascimento, State University of Campinas and EMBRAPA, Brazil
Keun H. Ryu, Chungbuk National University, Korea
Michael D. Soo, University of South Florida, USA
Andreas Steiner, TimeConsult, Switzerland
Vassilis Tsotras, Polytechnic University, USA
Jef Wijsen, Vrije Universiteit Brussel, Belgium

For additional information, see The TIMECENTER Homepage:
URL: `<http://www.cs.auc.dk/research/DBS/tdb/TimeCenter/>`

The TIMECENTER icon on the cover combines two "arrows." These "arrows" are letters in the so-called *Rune* alphabet used one millennium ago by the Vikings, as well as by their precedessors and successors. The Rune alphabet (second phase) has 16 letters, all of which have angular shapes and lack horizontal lines because the primary storage medium was wood. Runes may also be found on jewelry, tools, and weapons and were perceived by many as having magic, hidden powers.

The two Rune arrows in the icon denote "T" and "C," respectively.

Abstract

A wide range of real-world database applications, including financial and medical applications, are faced with accountability and trace-ability requirements. These requirements lead to the replacement of the usual update-in-place policy by an append-only policy, yielding so-called transaction-time databases. With logical deletions being implemented as insertions at the physical level, these databases retain all previously current states and are ever-growing. A variety of physical storage structures and indexing techniques as well as query languages have been proposed for transaction-time databases, but the support for physical deletion, termed vacuuming, has received precious little attention. Such vacuuming is called for by, e.g., the laws of many countries. Although necessary, with vacuuming, the database's previously perfect and reliable recollection of the past may be manipulated via, e.g., selective removal of records pertaining to past states. This paper provides a semantic framework for the vacuuming of transaction-time databases. The main focus is to establish a foundation for the correct and user-friendly processing of queries and updates against vacuumed databases. Queries that may return results affected by vacuuming are intercepted, and the user is presented with the option of issuing similar queries that are not affected by vacuuming.

Keywords: Vacuuming, physical deletion, transaction-time databases.

1 Introduction

Real-world database applications are frequently faced with accountability and trace-ability requirements, which in turn lead to so-called transaction-time databases that faithfully timestamp and retain all past states, thus offering their applications with a perfect, uncorrupted record of the past [Cop82].

However, these databases are also ever-growing, and business policies and legal laws demand the ability to physically delete data. Such physical deletion necessarily compromises irreversibly the previously perfect record of past states. It is thus a fundamental requirement to physical deletion capabilities that these be "controlled," which leads to the introduction of vacuuming.

We provide a framework that encompasses a range of new concepts essential to vacuuming. As part of this framework, we introduce vacuuming specifications and give their semantics. To meet the need for controlled physical deletion, vacuuming specifications include removal specification parts as well as so-called keep specification parts that override the former specification parts and specify what cannot be removed.

The paper explores how detection of potentially vacuuming-affected queries may be accomplished. Detection of queries that may yield results affected by vacuuming opens the possibility for taking cooperative action, providing, e.g., alternative queries that are similar to the original query, but are guaranteed to be unaffected by vacuuming. This detection is contingent on the disciplined modification of vacuuming specifications. Specifically, we introduce the notions of *growing* and *alive* specification parts, which open the possibility of vacuuming the vacuuming specifications without loosing track of what is removed by vacuuming.

The techniques proposed in the framework separate the enforcement of vacuuming semantics from the actual physical removal of data. This independence between correctness and physical vacuuming is highly desirable because it offers maximum flexibility for the scheduling of physical removal.

Only little work related to vacuuming has been published. A preliminary exploration of vacuuming was reported in reference [JM90]. In this unpublished technical report, we present different types of vacuuming specifications and introduce an algebra for defining vacuuming specifications. The present paper is based on and extends this report.

The TSQL2 temporal query language supports very basic vacuuming functionality: it is possible to specify cut-off points [SAA+94, Jen95] that indicate that data current only prior to a certain cut-off date

should be physically deleted. The semantic framework provided in this paper provides precise definitions of the concepts underlying this functionality, and provides much more advanced functionality.

Next, the Postgres DBMS [RS87], which supports transaction-time databases, includes a vacuuming cleaner daemon that is responsible for the asynchronous and transparent movement of logically deleted data from magnetic disk to cheaper optical disk storage. This kind of daemon and the associated techniques for physical deletion and reorganization and for the scheduling of the daemon may possibly be applied for implementing physical deletions. Beyond that, this research are accommodated by the framework presented here and is unrelated to this paper.

In the context of data warehousing, an approach to "expiring" data has recently been presented [GMLY98]. This work studies the removal of data not needed for maintaining predefined views. Further, all access to the data warehouse is assumed to occur through views. So, unlike in this paper, unrestricted, ad-hoc querying is not considered. Perhaps more importantly, because the underlying databases are not append-only, the correctness concerns fundamental to the work presented in this paper also play no role.

The contents are structured as follows. Section 2 offers an overview of a vacuuming-extended DBMS and identifies challenges posed by the introduction of vacuuming and met in the remainder of the paper. Section 3 provides the necessary details of the concrete data model context for the study of vacuuming, setting the stage for the introduction of vacuuming specifications, the semantics of which are defined in Section 4. Sections 5 and 6 then proceed to consider the querying and modification of databases with vacuuming, respectively. Notions of correctness and user-friendliness are the foci of these sections. Finally, Section 7 concludes and offers research directions.

2 Vacuuming—An Overview

In order to provide a global view of the paper's topic, this section gives a comprehensive example of a database system extended with vacuuming. Using the example, we introduce the semantics of vacuuming and consider the querying of vacuumed databases.

Assume we have an instance *emp* of a temporal relation schema *Emp*, given as follows.

$$Emp = \{S : SURROGATE; EmpId, Sal, Bal : INT; Sex : \{M, F\}; TT^{\vdash}, TT^{\dashv} : TIME\}$$

Attribute S is a tuple identifier, $EmpId$ identifies an employee, and Sal and Bal record the salaries and account balances of employees, respectively. Attributes $TT^{\vdash}$ and $TT^{\dashv}$ are both of type *TIME* and record the time of insertion and deletion, respectively[1].

First, we assume no vacuuming is specified on the database that the relation *emp* is part of. Then *emp* is given as in Table 1.

All updates of a temporal relation such as *emp* result in tuples being inserted, and tuples are never physically removed. (For emphasis, we will use "delete" for logical deletion and "remove" for physical removal throughout the paper.) Therefore, *emp* is ever-growing, and it is likely that *emp* will eventually contain some data that is irrelevant to its users or must be (physically) removed for other reasons. Now, assume that the current business policy is that data (logically) deleted more than four years ago is not to be retained, that tuples deleted between two and four years ago with value F of attribute Sex can be disregarded, but that all tuples in the database containing a Bal of $\$ - 5,000$ or less must be retained. Using standard relational algebra, this may be specified with the following vacuuming specification, $V = \{v_1, v_2, v_3\}$.

[1]Updates are modeled as deletions followed by insertions and assign deletion timestamps to some tuples and insertion timestamps to others.

S	$EmpId$	Sal	Bal	Sex	$TT^{\vdash}$	$TT^{\dashv}$
1	234	32k	$\$-6,015$	M	2/7/93	5/10/94
2	128	28k	$\$\ \ 10,274$	F	8/14/93	8/31/97
3	234	32k	$\$-2,015$	M	5/11/94	6/2/94
4	597	40k	$\$-4,652$	M	5/12/94	7/2/94
5	597	47k	$\$-2,576$	M	7/3/94	NOW
6	234	35k	$\$\ \ 1,763$	M	11/8/94	NOW
7	318	21k	$\$\ \ 211$	F	11/24/94	6/2/95

Table 1: The *emp* Relation

$$
\begin{aligned}
v_1 \quad & \rho(emp) : \sigma_{TT^{\dashv} \leq NOW-4yrs}(emp) \\
v_2 \quad & \rho(emp) : \sigma_{NOW-4yrs<TT^{\dashv} \leq NOW-2yrs \wedge Sex=F}(emp) \\
v_3 \quad & \kappa(emp) : \sigma_{Bal \leq \$-5,000}(emp)
\end{aligned}
$$

The specification is read as follows: "Remove (ρ) from *emp* all tuples deleted more than four years ago, i.e., tuples where the value of the attribute $TT^{\dashv}$ is less than the current time, NOW, minus four years. Remove from *emp* all tuples deleted between two and four years ago where attribute Sex has value F. Keep (κ) in *emp* all tuples where attribute Bal has the value $\$-5,000$ or less."

While v_1 and v_2 are removal specification parts and tell what possibly can be removed, v_3 is a keep specification part stating what must be kept. Keep specification parts always override removal specification parts, for safeguarding reasons. Submitting specification V yields the vacuumed relation *emp* in Table 2, the current time (and the value of variable NOW [CDI$^+$97]) being $7/14/98$.

S	$EmpId$	Sal	Bal	Sex	$TT^{\vdash}$	$TT^{\dashv}$
1	234	32k	$\$-6,015$	M	2/7/93	5/10/94
2	128	28k	$\$\ \ 10,274$	F	8/14/93	8/31/97
5	597	47k	$\$-2,576$	M	7/3/94	NOW
6	234	35k	$\$\ \ 1,763$	M	11/8/94	NOW

Table 2: Relation *emp* at Time $7/14/98$, Vacuumed According to $V = \{v_1, v_2, v_3\}$

Without vacuuming, a transaction-time relation satisfies the property of *faithful history encoding* (to be formalized later), stating that previously current database states are retained. This property is jeopardized when vacuuming is allowed. To illustrate this, an example follows showing that querying a vacuumed database, it is possible to obtain answers affected by vacuuming, i.e., some queries return different answers than they would have without vacuuming.

Assume the query $Q = \sigma_{Sal \geq 35k}(emp)$ is issued, let $V = \emptyset$, and assume that the current time is $7/14/98$. Then Q would evaluate to the result shown in Table 3. If instead $V = \{v_1, v_2, v_3\}$, Q will evaluate to tuples 5 and 6, since tuple 4 was logically deleted more than four years ago and has $Bal > \$-5,000$.

So query Q is affected by the vacuuming according to V, and in general $Q(emp, \emptyset) \neq Q(emp, V)$ (here, (emp, V) denotes relation *emp* vacuumed according to V). Thus, our sample query Q returns a result inconsistent with the previously current database states. This result is misleading to users expecting faithful history encoding. Users knowing that faithful history encoding may have been compromised are unable to properly interpret the answer. To properly interpret an answer, the users must understand the vacuuming specifications in effect.

S	$EmpId$	Sal	Bal	Sex	$TT^{\vdash}$	$TT^{\dashv}$
4	597	40k	$\$ -4,652$	M	5/12/94	7/2/94
5	597	47k	$\$ -2,576$	M	7/3/94	NOW
6	234	35k	$\$ \quad 1,763$	M	11/8/94	NOW

Table 3: The Result of $Q = \sigma_{Sal \geq 35k}(emp, \emptyset)$

A system with vacuuming should support its users in interpreting the results of queries. Specifically, the system should support *faithful history querying* (also to be formalized later), stating that only queries unaffected by vacuuming are answered without an accompanying warning. On the other hand, queries that may return results affected by vacuuming must return an error or be accompanied by a warning. Assuming that an error message is returned, the system should return also at least one alternative, similar query, Q', satisfying faithful history querying. This is illustrated in Figure 1. The reader may verify that the alternative

```
>> select [Sal >= 35k] (emp)
  Error:  Query affected by vacuuming; alternative query:
  select [((TTend > NOW - 4yrs and (TTend > NOW - 2yrs or Sex = M))
    or Bal <= -5000) and Sal >= 35k] (emp)
  Run?  (Y)
```

Figure 1: The Result of Submitting Query $Q = \sigma_{Sal \geq 35k}(emp, V)$

query (Q') returns the tuples $\{5, 6\}$. So, issued on the vacuumed version of *emp*, Q' returns the same answer as Q (see Tables 1 and 2). However, query Q' is not affected by vacuuming, i.e., $Q'(emp, \emptyset) = Q'(emp, V)$, and this query thus satisfies the faithful history querying property.

Presented with the vacuuming-modified query, the user can choose either to issue this query or to modify it and reissue the result. In the latter case, the system may have to go through a new modification-and-display process.

In the next sections, we present the necessary parts of a data model that will serve as the context for introducing vacuuming; we show how to determine whether a query such as Q satisfies faithful history querying and how to determine a similar query that is not affected; and we consider the modification of the vacuuming of user-defined relations and vacuuming specifications.

3 Data Model Context

A concrete data model context is needed for presenting the vacuuming framework. This section presents the necessary aspects of the temporal data model that provides the context for our study. Initially, the data structures, schemas as well as instances, of the model are presented. Then the syntax of vacuuming specifications is given. The framework is independent of the particular query language adopted, so rather than adopting one of the many existing temporal algebras or defining yet a new algebra [MS91], we reuse the well-known relational algebra as the language associated with the data structures.

3.1 Temporal Relation Structures

Let $U_D = \{D_1, D_2, \ldots, D_D\}$ be a set of non-empty domains, and let $D = \cup D_i$ be the set of all values. Let $D_V = \{v_1, v_2, \ldots, v_V\}$ be the specific domain of vacuuming specification parts. Let $T = \{t_0, t_1, \ldots, t_x\}$

be a finite, non-empty set of times with $<$ as the total order relation. We use element t_{now} in T for denoting the current time. Finally, let $T_{NOW} = T \cup \{NOW\}$, where NOW is a variable that evaluates to the current time [CDI+97]. Then, for $t \in T$ and $t' \in T_{NOW}$, we define the meaning of t' at time t, $[\![t']\!]_t$, as follows.

$$[\![t']\!]_t = \begin{cases} t & \text{if } t' = NOW \\ t' & \text{otherwise} \end{cases}$$

Next, let $U_A = \{A_1, A_2, \ldots, A_A\}$ be a set of attributes, let $Vspec \in U_A$, and let $TT^{\vdash}$ and $TT^{\dashv}$ be distinguished time attributes representing insertion and deletion time, respectively [SA85]. With these definitions in place, we can define the schema aspects of a database.

A *temporal relation schema*, $\mathcal{R}_x$, is defined as a pair $\langle A_{R_x}, DOM_{R_x}\rangle$, where: (1) $A_{R_x} \subseteq U_A$, and $A_{R_x} \cup \{TT^{\vdash}, TT^{\dashv}\}$ is the set of attributes of the schema. The latter two attributes are timestamp attributes of R_x; (2) DOM_{R_x} is a function from $A_{R_x} \cup \{TT^{\vdash}, TT^{\dashv}\}$ to $U_D \cup \{T, T_{NOW}\}$, which assigns domains in U_D to attributes in A_{R_x}, the domain T to the attribute $TT^{\vdash}$, and the domain T_{NOW} to the attribute $TT^{\dashv}$.

Next, symbol $\mathcal{V}$ denotes the specific temporal relation schema $\langle\{Vspec\}, DOM_V\rangle$ for vacuuming specification parts, where DOM_V is a function assigning the domains D_V, T, and T_{NOW} to the attributes $Vspec, TT^{\vdash}$, and $TT^{\dashv}$, respectively.

A *temporal database schema* $\mathcal{DB}$ is then a finite set of temporal relation schemas $\mathcal{R}_x = \langle A_{R_x}, DOM_{R_x}\rangle$, one of them being the schema $\mathcal{V} = \langle\{Vspec\}, DOM_V\rangle$.

EXAMPLE: In Section 2, *emp* is the temporal relation with schema $\langle A_{emp}, DOM_{emp}\rangle$, where $A_{emp} = \{S, EmpId, Sal, Bal, Sex\}$ and DOM_{emp} is the function assigning the domain *SURROGATE* to S, the domain *INT* to each attributes $EmpId$, Sal, and Bal, the domain $\{M, F\}$ to attribute Sex, and finally the domains T and T_{NOW} to $TT^{\vdash}$ and $TT^{\dashv}$, respectively.

Furthermore, we have $\mathcal{DB} = \{\langle A_{emp}, DOM_{emp}\rangle, \ldots, \langle\{Vspec\}, DOM_V\rangle\}$, as an example of a temporal database schema. □

We proceed to define instances of the schemas just defined. A *tuple*, u, on relation schema $\langle A_{R_x}, DOM_{R_x}\rangle$ is a function from the attribute set $A_{R_x} \cup \{TT^{\vdash}, TT^{\dashv}\}$ to $D \cup \{T, T_{NOW}\}$, which assigns an element in $DOM_{R_x}(A_i)$ to each attribute $A_i \in A_{R_x}$, an element in T to $TT^{\vdash}$, and an element in T_{NOW} to $TT^{\dashv}$; u assigns the elements to the satisfaction of:

$$\forall t' \geq u.TT^{\vdash}\ (u.TT^{\vdash} \leq [\![u.TT^{\dashv}]\!]_{t'} \wedge (u.TT^{\vdash} = t_{now} \Rightarrow u.TT^{\dashv} = NOW)).$$

This formula simply states that intervals must start no later than when they end and that the end time must be NOW if the start time is the current time.

For any pair of tuples u_1 and u_2 on relations with the same explicit attributes A_i, we say that u_1 and u_2 are *value equivalent*, $u_1 \overset{v.e.}{=} u_2$, if and only if $\forall A_i\ (u_1.A_i = u_2.A_i)$.

We also say that a tuple u on relation R_x is *current at time* t in the database if and only if $u.TT^{\vdash} \leq t \leq [\![u.TT^{\dashv}]\!]_t$, and more specifically that a tuple is *current* in the database if it is current at t_{now}.

We define a *temporal vacuuming specification part*, v, to be a tuple on the vacuuming specification schema $\langle\{Vspec\}, DOM_V\rangle$, assigning values from the domain D_V to the $Vspec$ attribute, and values from T and T_{NOW} to $TT^{\vdash}$ and $TT^{\dashv}$, respectively.

With tuples in place, we proceed to relations. A *temporal relation* R_x with schema $\langle A_{R_x}, DOM_{R_x}\rangle$ is a finite set of tuples. R_x does not contain value equivalent tuples that are current at the same time. We term these user-defined relations. As a specific temporal relation, we define a *temporal vacuuming specification* V with schema $\mathcal{V} = \langle\{Vspec\}, DOM_V\rangle$. Thus, V is a finite set of temporal vacuuming specification parts.

Having a temporal relation R_x *and* a vacuuming specification V, the effect of specifying vacuuming for R_x is a modified relation $\hat{R}_x$, written as (R_x, V). So $\hat{R}_x$ is the relation R_x modified by the vacuuming specification V. We will return to the semantics of vacuuming, and to the definition of $\hat{R}_x$ in Section 4.

Finally, a *temporal database* DB with schema $\mathcal{DB} = \{\mathcal{R}_1, \ldots, \mathcal{R}_n, \mathcal{V}\}$ is a set of temporal relations modified by vacuuming specification V, i.e., $DB = \{\hat{R}_1, \ldots, \hat{R}_n, \hat{V}\}$.

EXAMPLE: Our sample database contains the modified temporal relations $\hat{emp}$ and $\hat{V}$.

Relation *emp* contains a set of tuples, and as shown in Table 1, the first tuple is the function assigning value 1 to S, values 234, $32k$, $\$-6,015$ to $EmpId$, Sal, Bal, respectively, value M to Sex, and transaction timestamps 2/7/93 and 5/10/94 to the time attributes $TT^{\vdash}$ and $TT^{\dashv}$, respectively.

The temporal vacuuming specification $V = \{v_1, v_2, v_3\}$ shown in Section 2 is given next as the relation V.

	$Vspec$		$TT^{\vdash}$	$TT^{\dashv}$
v_1	$\rho(emp)$	$: \sigma_{TT^{\dashv} \leq NOW-4yrs}(emp)$	$5/16/1992$	NOW
v_2	$\rho(emp)$	$: \sigma_{NOW-4yrs<TT^{\dashv} \leq NOW-2yrs \wedge Sex=F}(emp)$	$8/30/1995$	NOW
v_3	$\kappa(emp)$	$: \sigma_{Bal \leq \$-5,000}(emp)$	$5/16/1992$	NOW

Here, v_1 is the tuple that assigns the *STRING* value "$\rho(emp) : \sigma_{TT^{\dashv} \leq NOW-4yrs}(emp)$" to attribute $Vspec$, the time 5/16/1992 to $TT^{\vdash}$, and the variable NOW to $TT^{\dashv}$.

Later, after defining the semantics of vacuuming, we will return to the modified counterparts of *emp* and V. For now, Tables 1 and 2 show relation *emp* and its modified counterpart $\hat{emp}$, respectively. □

3.2 Syntax of Vacuuming Specifications

We have defined a vacuuming specification part as a tuple on the schema $\langle \{Vspec\}, DOM_V \rangle$, assigning values from D_V to $Vspec$. Next, we specify which values domain D_V offers as possible, or *well-formed*, specification part expressions.

One aspect of a specification part expression being well-formed is being syntactically correct. The syntax of a vacuuming specification part expression v is given by the following specification. Note that this specification essentially permits arbitrary relational algebra selections.

$$
\begin{array}{lll}
v & ::= & \omega(R) : Exp \\
\omega & ::= & \rho \mid \kappa \\
Exp & ::= & R \mid \sigma_F(Exp) \mid (Exp) \\
F & ::= & \texttt{true} \mid \texttt{false} \mid F\ bop\ F \mid \neg F \mid (F) \mid TT\ op\ tt \mid tt\ op\ TT \mid d\ op\ A_i \mid A_i\ op\ d \\
tt & ::= & t \mid NOW \mid tt - tt \mid tt + tt \mid (tt) \\
TT & ::= & TT^{\vdash} \mid TT^{\dashv} \\
bop & ::= & \vee \mid \wedge \\
op & ::= & < \mid > \mid = \mid \leq \mid \geq \mid \neq
\end{array}
$$

In addition to being syntactically correct, a specification part expression must also satisfy conventional semantic constraints. It is required that $d \in D_{A_i}$, $A_i \in U_A$, and $t \in T$ in the specification above. For expressions such as $A_i\ op\ d$ and $d\ op\ A_i$, op should be defined for the domain D_{A_i} of A_i and $d \in D_{A_i}$. And for $\sigma_F(Exp)$, F should only include attributes A_i in Exp.

Note that any specification part expression defined here can be rewritten to be on the form "$\omega(R_x) : \sigma_P(R_x)$," using standard equivalence-preserving transformations (e.g., [ASU79b, Ull88a]).

Specification parts having expressions of the form "$\rho(R_x) : Exp$" are *removal specification parts*, and specification parts having expressions of the form "$\kappa(R_x) : Exp$" are *keep specification parts*. From our example, v_1 and v_2 are removal specification parts, and v_3 is a keep specification part.

4 Specification Semantics

Having defined a data model context for vacuuming and its syntax, we turn to defining the semantics of a vacuuming specification, V. The semantics of V expresses for each relation R_x in the database what remains in R_x with V in effect. This expression was previously denoted by $\hat{R}_x$ and is termed the *modified relation*. In defining the modified relation, three issues are considered. First we consider the objectives to be satisfied by the definition of the semantics. The second issue is how to take into account the pairs of time values associated with vacuuming specification parts and how to properly account for NOW-relative specification parts in the semantics. Third, we define the modified relation itself.

Considering the first issue, the definition of the meaning of a vacuuming specification aims at satisfying two objectives, namely ease of use and loss protection. The rationale for the former objective is self-evident; the latter is important because vacuuming is irreversible. It should thus be possible to guard against unintended removal of data.

With these objectives in mind, the semantics of a vacuuming specification will be defined to be independent of the insertion order of the vacuuming specification parts. This is consistent with the formalization of a vacuuming specification as a relation.

Mainly to guard against unintended removal of data, but also to provide increased ease of use, we have included keep specification parts that specify what must be kept in the database.

Wanting to express—without the use of keep specification parts—that certain tuples from a relation are to be retained in the database can be done by making sure that no removal specification part selects these tuples for removal. But this is only an implicit expression, making it difficult to maintain. Further, it does not protect against unintended loss of data. With keep specification parts that override the removal specification parts, it is instead possible to specify in a single specification part the tuples that are to be retained. This is easier, and new removal specification parts are guaranteed to not inadvertently lead to the removal of tuples to be kept. In general, specifications may become simpler with both keep and removal specification parts available.

Having both keep and removal specification parts and an order-independent semantics, where the keep specification parts override the removal specification parts, improves ease of use and facilitates loss protection. We thus define vacuuming in this way.

With the semantics decided upon at this abstract level, we turn to the second issue of determining how the vacuuming specification parts contribute to vacuuming based on their temporal aspects. Being a temporal relation, each part of a specification is timestamped with $TT^{\vdash}$ and $TT^{\dashv}$ values that indicate when the part was inserted and subsequently logically deleted. How should these time values be taken into account in the semantics?

A first thought may be that only current vacuuming specification parts should be taken into account in the semantics of a specification. However, the semantics must express for each relation what is left in the relation, independently of when the missing data was removed. Because even non-current (logically deleted) vacuuming specification parts may be responsible for the absence of tuples from a relation, all parts, non-current as well as current, must be taken into account in the semantics.

However, the $TT^{\vdash}$ and $TT^{\dashv}$ values of a part do affect the semantics. Recall that vacuuming specification parts may involve the variable NOW that evaluates to the current time, making them NOW-relative. For example, specification part v_2 in the running example specifies the removal of tuples from relation *emp* with $Sex = F$, for which $NOW - 4yrs < TT^{\dashv} \leq NOW - 2yrs$. This specification part was inserted on 8/30/1995 and remains current. So at current time the effect of this specification is the removal of tuples that at some time between 8/30/1995 and the current time have satisfied the specified property, that is tuples with $Sex = F$ and for which $\exists t\ (t - 4yrs < TT^{\dashv} \leq t - 2yrs \wedge \text{8/30/1995} \leq t \leq t_{now})$.

In general, the expression of a vacuuming specification part is modified to take into account its timestamps as follows. All occurrences of NOW in the expression are replaced by an unused variable t, the

expression is augmented by the term "$\wedge\ TT^{\vdash} \leq t \leq [\![TT^{\dashv}]\!]_{t_{now}}$," and the resulting expression is existentially quantified by t. Equivalence-preserving transformations may subsequently be applied to the modified specification parts in order to simplify them. Recalling that each specification part can be rewritten in the form "$\omega(R_x) : \sigma_P(R_x)$," modifying a specification part to take its timestamps into account gives a specification part in the form "$\omega(R_x) : \sigma_{\exists t\ (P' \wedge TT^{\vdash} \leq t \leq [\![TT^{\dashv}]\!]_{t_{now}})}(R_x)$," where P' is the predicate P with NOW replaced by t.

Modifying this way any user-specified vacuuming specification part that follows the syntax defined in Section 3.2 yields a well-defined expression specifying in *constant terms* exactly what is selected by a NOW-relative specification part from its insertion until the current time. Note that a logical deletion (which corresponds to replacing value NOW of attribute $TT^{\dashv}$ by a fixed value) fixes the upper bound of the vacuuming to some time before the current time, t_{now}.

EXAMPLE: The selection predicate in specification part v_2 (with $TT^{\vdash}$ = 8/30/1995 and $TT^{\dashv} = NOW$) modified as explained above may be simplified as described next. We assume that the current time t_{now} is 7/14/1998.

$$\begin{aligned}
\exists t\ \ & (Sex = F\ \wedge\ t - 4yrs < TT^{\dashv} \leq t - 2yrs\ \wedge\ \text{8/30/1995} \leq t \leq t_{now}\ \wedge\ t_{now} = \text{7/14/1998}) \\
= \ & (Sex = F\ \wedge\ \text{8/30/1995} - 4yrs < TT^{\dashv} \leq t_{now} - 2yrs\ \wedge\ t_{now} = \text{7/14/1998}) \\
= \ & (Sex = F\ \wedge\ \text{8/30/1991} < TT^{\dashv} \leq \text{7/14/1996})
\end{aligned}$$

□

In addition to NOW-relative specification parts, parts that specify vacuuming in the future are meaningful and thus allowed, although they do not appear to be very useful. To understand the issue, consider a removal specification part with predicate "$NOW - 1yrs \leq TT^{\dashv} \leq NOW + 1yrs$." When this part is deleted, at some time t_{now}, the upper bound of $TT^{\dashv}$ of tuples to be removed is $t_{now} + 1yrs$, i.e., one year into the future. So for one year after having deleted the specification part, this part continues to remove tuples; the deletion does not stop the removals. The removal specification part with predicate "$NOW - 1yrs \leq TT^{\dashv}$" would remove exactly the same tuples as the first one above, during the time both are current, and it ceases to remove tuples when logically deleted. Note that both specification parts result in tuples being removed immediately upon insertion. This kind of inappropriate specifications are expected to be rejected by an implementation of vacuuming.

Having considered the various issues, we are able to define the semantics of a vacuuming specification in terms of its effect on each relation in turn. To do that, we define $V|_{R_x}$ as all specification parts in V that concern R_x, i.e., parts with a $Vspec$ value of the form "$\omega(R_x) : Exp$." Now, let $V|_{R_x} = \{v_1, \ldots, v_k, v_{k+1}, \ldots, v_s\}$, where $v_i \in \{v_1, \ldots, v_k\}$ are *removal* specification parts and $v_j \in \{v_{k+1}, \ldots, v_s\}$ are *keep* specification parts. Following the observation in Section 3.2 and taking the timestamps into account as above, all v_i's specifying vacuuming for a relation R_x can be reduced to the form "$\omega(R_x) : \sigma_{F_i}(R_x)$," where F_i is of the form "$\exists t_i\ (P_i' \wedge TT^{\vdash} \leq t_i \leq [\![TT^{\dashv}]\!]_{t_{now}})$." We assume without loss of generality that each v_i and v_j above are of this form.

Holding the assumptions and notation introduced above in mind, we define the modified relation of R_x at current time, $\hat{R}_x$, as follows.

$$\hat{R}_x \quad \stackrel{def}{=} \quad \sigma_{(\neg \bigvee_{i=1}^{k} F_i) \vee (\bigvee_{j=k+1}^{s} F_j)}(R_x) \tag{1}$$

So, the modified relation $\hat{R}_x$ is the set of tuples from R_x either not satisfying any predicate of a removal specification part F_i, or, if so, also satisfying the predicate of at least one keep specification part F_j. Tuples not satisfying any predicates at all are present in $\hat{R}_x$, and so are tuples satisfying any number of predicates from keep specification parts. Also, no tuples satisfying only predicates from removal specification parts are present in $\hat{R}_x$. This way, keep specification parts override removal specification parts.

EXAMPLE: Let us illustrate vacuuming by creating the expression of the modified relation $\hat{emp}$, from relations *emp* and $V = \{v_1, v_2, v_3\}$ presented in Section 2. V contain only well-formed specification parts, which by equivalence transformations can be rewritten to be on the form "$\omega(R_x) : \sigma_P(R_x)$." Since v_1 and v_2 are NOW-relative specifications, they are rewritten to the form "$\omega(R_x) : \sigma_{\exists t\ (P' \wedge TT^{\vdash} \leq t \leq [\![TT^{\dashv}]\!]_{t_{now}})}(R_x)$." Now, from Equation 1 we get the modified relation $\hat{emp} = (emp, V) = (\sigma_{\neg(F_1 \vee F_2) \vee F_3}(emp), \emptyset) = (\hat{emp}, \emptyset)$. Let F_1, F_2, and F_3 be the selection predicate in the rewritten specification parts v_1, v_2, and v_3, respectively. Then the selection predicate $F' = \neg(F_1 \vee F_2) \vee F_3$ will be:

$$\begin{aligned} F' &= \neg\,[\exists t_1\ (TT^{\dashv} \leq t_1 - 4yrs\ \wedge\ \text{5/16/1992} \leq t_1 \leq t_{now})\ \vee \\ &\quad\ \exists t_2\ (t_2 - 4yrs < TT^{\dashv} \leq t_2 - 2yrs\ \wedge\ \text{8/30/1995} \leq t_2 \leq t_{now}\ \wedge\ Sex = F)] \\ &\quad \vee\ [Bal \leq \$ - 5,000] \\ &= \neg\,[TT^{\dashv} \leq t_{now} - 4yrs\ \vee\ (\text{8/30/1991} < TT^{\dashv} \leq t_{now} - 2yrs\ \wedge\ Sex = F)] \\ &\quad \vee\ [Bal \leq \$ - 5,000] \end{aligned}$$

Note that the vacuuming-modified relation $\hat{emp}$ can be vacuumed due to V without changing; no additional tuples will be kept or removed. Finding the vacuuming-modified expression, when vacuuming one more time, is done using the selection predicates in the same way, and since they are already present in the first vacuuming-modified expression, they can be left out leaving the same vacuuming-modified expression. In our example this gives:

$$\begin{aligned} \big(\sigma_{\neg(F_1 \vee F_2) \vee F_3}(emp), V\big) &= \big(\sigma_{\neg(F_1 \vee F_2) \vee F_3}(\sigma_{\neg(F_1 \vee F_2) \vee F_3}(emp)), \emptyset\big) \\ &= \big(\sigma_{\neg(F_1 \vee F_2) \vee F_3}(emp), \emptyset\big) \end{aligned}$$ □

A system that implements vacuuming should obey the semantics defined above. On the other hand, it is also attractive for the system to not have to actually perform physical removals eagerly to ensure that the semantics are obeyed. Rather, lazy physical removal is attractive.

In order to both ensure correct semantics and permit lazy removal, the system may use the vacuuming-modified relation expressions defined above in place of the corresponding relations themselves. The expressions then serve as shields that hide the tuples in the relations that have been vacuumed logically, but may or may not yet have been physically removed.

5 Querying Vacuumed Databases

Having defined the notion of a database system with vacuuming facilities, we now turn to the querying of databases in the context of vacuuming. Transaction-time databases without vacuuming retain a perfect record of past states. When introducing vacuuming, this is no longer guaranteed, and the results of queries become harder to interpret. For example, a query on a past state may return an empty result either because this state never contained qualifying data or because all qualifying data has been removed because of vacuumed. In this section, we specify the property of faithful history querying, aiming at making queries on vacuumed databases easier to interpret. Next, we lay out a strategy for satisfying this property, and we present the details of the solution. Throughout we use the example introduced in Section 2 for illustration.

5.1 Faithful History Encoding and Querying

A transaction-time database without vacuuming retains all previously current states. So a query that retrieves the current database state at some time t, and the query that at some later time retrieves the database state recorded as being current at time t, will both give the same result. This property, we term *faithful history encoding*.

To give a precise definition, we need to define the meaning of retrieving the state current as of some time. For this purpose, we define the timeslice $\tau_t(R_x)$ of R_x at time t [Sch77]. This operator returns a non-temporal relation having the explicit attributes of R_x. The relation contains the set of tuples that are value equivalent to the tuples in relation R_x current at time t.

$$\tau_t(R_x) = \{u \mid u' \in R_x \wedge u \stackrel{v.e.}{=} u' \wedge u'.TT^{\vdash} \leq t \leq [\![u'.TT^{\dashv}]\!]_t\}$$

To define *faithful history encoding*, we also need to be able to "rollback" a relation to how it was at some past time.

For this purpose, let $[\![R_x]\!]_t$ denote relation R_x at time t, i.e., the set of tuples present in the relation at this time. Then $[\![R_x]\!]_t$ contains the set of tuples inserted into the relation before or at time t, even if they were later deleted (i.e., deleted between time t and the current time); further the timestamps of the resulting tuples are restored to their original appearance at time t. More formally, $[\![R_x]\!]_t$ is defined as follows.

$$\begin{aligned}[\![R_x]\!]_t \stackrel{def}{=} \{u \mid \exists u' \in R_x\ (u \stackrel{v.e.}{=} u' \wedge u.TT^{\vdash} = u'.TT^{\vdash} \wedge u.TT^{\vdash} \leq t \wedge \\ ((u.TT^{\dashv} = u'.TT^{\dashv} \wedge [\![u'.TT^{\dashv}]\!]_t \leq t) \vee \\ (u.TT^{\dashv} = NOW \wedge [\![u'.TT^{\dashv}]\!]_t > t)))\}\end{aligned}$$

So to obtain the result, we first consider only the subset of R_x inserted no later than time t. If a tuple was deleted after time t, we replace the deletion time with the value it actually had at time t, NOW; otherwise, the tuples from our subset are returned unmodified. Note that $[\![R_x]\!]_{t_{now}} = R_x$.

EXAMPLE: To illustrate the definition, consider relation *emp* in Table 1. $[\![emp]\!]_{10/1/94}$ denotes the set of tuples shown in Table 4. Tuples 6 and 7 where inserted in *emp* after $10/1/94$, so they are not present

S	$EmpId$	Sal	Bal	Sex	$TT^{\vdash}$	$TT^{\dashv}$
1	234	32k	$\$-6,015$	M	2/7/93	5/10/94
2	128	28k	$\$\ 10,274$	F	8/14/93	NOW
3	234	32k	$\$-2,015$	M	5/11/94	6/2/94
4	597	40k	$\$-4,652$	M	5/12/94	7/2/94
5	597	47k	$\$-2,576$	M	7/3/94	NOW

Table 4: The $[\![emp]\!]_{10/1/94}$ Relation

here. Tuple 2 had a transaction-time end of $8/31/97$ and $[\![8/31/97]\!]_{10/1/94} = 8/31/97$, which is larger that $10/1/94$. Thus tuple 2 receives the variable NOW as its new transaction-time end value. Tuples 1, 3, 4, and 5 all have $[\![TT^{\dashv}]\!]_{10/1/04} \leq 10/1/94$, so they retain their transaction-time end values. The tuples in the table are exactly the tuples in *emp* at time $10/1/94$. □

With the two preceeding definitions, we can precisely define *faithful history encoding*.

$$\forall R_x \big(\forall t \leq t_{now}\ \big(\tau_t(R_x) = \tau_t([\![R_x]\!]_t)\big)\big) \tag{2}$$

That is, for all relations and all times t not exceeding the current time, evaluating the timeslice with time parameter t on the relation as it was at time t versus on the current relation gives the same result. As a result, all previously current states are retained.

With faithful history encoding, if a query on a past state returns an empty result then this means that there never were qualifying tuples in this state. However, transaction-time databases with vacuuming are unable to satisfy the property of faithful history encoding, and this inference cannot be made.

Rather than simply giving no guarantees, we instead give a guarantee, termed *faithful history querying*, that attempts to get as close to faithful history encoding as possible. This new property states that only queries that return the same answers when submitted to the vacuumed database as when submitted to the corresponding unvacuumed one should be answered. With this property satisfied, misinterpretations of the answers are prevented.

Formulated precisely, the property of *faithful history querying* is satisfied if and only if the following holds for all queries Q that are answered by the system.

$$\forall R_x \ \big(Q(R_x, V) = Q(R_x, \emptyset)\big) \tag{3}$$

EXAMPLE: To illustrate faithful history querying, we consider two sample queries based on the running example.

The first query, $Q_1 = \sigma_{TT^{\dashv}=NOW \wedge Bal \geq \$0}(emp)$, only selects from the part of relation *emp* not affected by vacuuming. Therefore, it is unaffected by vacuuming, and a system satisfying faithful history querying needs not reject this query, but can give the answer, in this case $\{6\}$.

The second query, $Q_2 = \sigma_{Sal \geq 35k}(emp)$, overlaps with the part of *emp* affected by vacuuming. With this query, it is impossible to say whether the answer will be correct or not, and the system must for safety reasons deny answering this query. □

Having defined the desirable *faithful history querying* property, the next step is to consider how to achieve a system that supports this property. It is essential for the system to be able to detect potential problematic queries.

5.2 Query Handling

A vacuuming-enhanced system that satisfies *faithful history querying* answers only some queries, while taking other actions for the remaining queries. The four overall steps necessary to achieve this functionality are as outlined next and are discussed in detail in the remainder of this section.

1. At vacuuming specification time, create expressions for the vacuuming-modified relations.
2. At query time, create the modified counterpart of the query submitted, obtained by replacing the relation names in the query with the corresponding vacuuming-modified expressions for the relations.
3. Check if the modified and the original queries are equivalent. If yes, the original query is not affected by vacuuming.
4. If the result of the previous step is affirmative, the query is evaluated and the answer is returned. The possible responses to the other outcome will be discussed later in this section.

The first step is to create the vacuuming-modified relation as an expression on the unvacuumed relation. This was taken care of in Section 4.

EXAMPLE: In Section 4, we obtained the following expression $\sigma_{F'}(emp)$ for the modified version of relation *emp*. F' is given by

$$\neg\ [(TT^{\dashv} \leq t_{now} - 4yrs) \vee (8/30/1991 < TT^{\dashv} \leq t_{now} - 2yrs\ \wedge\ Sex = F)] \vee [Bal \leq \$ - 5,000],$$

where t_{now} denotes the current time. □

The second step occurs when a query Q is issued and the system must test whether the query violates faithful history querying. In this step, a vacuuming-modified version Q' of Q is created. This modified version is obtained replacing all relation names in Q by the expressions for the corresponding vacuuming-modified relations. The technique used here is known as *query modification* and is the technique traditionally used for implementing integrity constraints and views [Sto75]. 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.

In the third step, an equivalence test is performed on Q and Q'. Although it has been shown that the general problem of determining equivalence of relational expressions is NP-complete, efficient algorithms have also been devised for determining equivalence for an important subset of relational expressions (most practical SPJ-queries) [ASU79a, ASU79b, PS88]. So, the test employed is one that will never succeed if, in fact, Q and Q' are not equivalent (soundness), but also one that 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.

EXAMPLE: In the second example in Section 5.1, we considered two queries. The first was $Q_1 = \sigma_{TT^{\dashv}=NOW \wedge Bal \geq \$0}(emp)$. When this query is issued, we replace *emp* with the expression $\sigma_{F'}(emp)$ given in the previous example to obtain the modified version, Q'_1. Using standard equivalence transformations, it is straightforward to verify that the original and modified queries are equivalent, $Q_1 \equiv Q'_1$. (Note that occurrences of NOW in a query are replaced by t_{now} when it is issued to the system.) The system can therefore evaluate Q_1 and return the answer without violating faithful history querying.

The second query was $Q_2 = \sigma_{Sal \geq 35k}(emp)$. It is easy to see that this query is not equivalent to $Q_2 = \sigma_{Sal \geq 35k}(\sigma_{F'}(emp))$, again using the definition of F' given in the previous example. It will thus constitute a violation of *faithful history querying* to return an answer for query Q_2. □

The fourth step remains. If the outcome of the equivalence test is positive, the system proceeds to evaluate query Q and returns the answer to the user. The result is unaffected by any vacuuming and by whether any parts of the database selected for removal still remain in the database, e.g., because the system uses a lazy policy for the physical removal.

If the outcome of the test is negative, the system should not simply evaluate query Q. Some other action should be taken. In the remainder, we explore the options.

Focussing first on application access, the natural approach would be giving an error or a warning. An error message might be accompanied by reasons for the error and perhaps by alternative queries that do satisfy faithful history querying. A warning might be accompanied by the answer to the query along with alternative queries and reasons as well. The application can then use exception-handling, and, depending on the warning codes, choose how to proceed.

In an interactive situation, the preferences are a bit different. Some users may be closely familiar with the vacuuming specifications and may want answers even to queries that violate faithful history querying. Violation of faithful history querying might be preferable for these users. Other users might expect answers that satisfy faithful history querying and may thus misinterpret answers that do not. For these, receiving an error message and one or more alternative queries would be appropriate.

Common to the situations is the presentation of one or more alternative queries. But what alternative queries should be given to the user? One possibility is to return the vacuuming-modified query expression Q' to the user. Given the vacuuming in effect, this query is as similar to Q as possible while also satisfying the required condition of being equivalent to its own vacuuming-modified version, i.e., satisfying $Q' \equiv Q''$. It is possible to apply equivalence-preserving transformations to expression Q' with the purpose of simplifying it before returning it to the user. For example, the transformations also used during query optimization (e.g., [SC75, Ull88b]) are applicable. This was the option chosen in Section 2 (recall Figure 1).

As a further extension, several alternative queries can be derived using techniques for *query generalization and specialization* [Mot84, Cha90]. Specialized versions of Q' are more restrictive than Q' and return a smaller result than Q'. Such queries do thus not violate faithful history querying. Generalized versions of Q' are less restrictive than Q' and return larger results than Q'. To make sure that these do not violate faithful history querying, they must either be constructed carefully, or they should be subjected to and pass the equivalence test prior to being presented to the user.

6 Modifying Vacuumed Databases

Having defined vacuuming and having also covered the querying of vacuumed databases, database modification remains to be covered. We may distinguish between the four cases obtained by combining (i) regular modification versus (ii) vacuuming with (a) regular user-defined relations versus (b) the special relation V that has vacuuming specification parts as tuples. The introduction of vacuuming poses no constraints on modifications—insertions, updates, and deletions—of regular user-defined relations, leaving three cases. Because the vacuuming of regular user-defined relations and of the vacuuming relation are achieved by modifying relation V, the remaining three cases all reduce to modification of V.

Section 6.1 covers modification of relation V, considering vacuuming specification parts of the form "$\omega(R_x) : Exp$" (see Section 3.2), where R_x is any relation, user-defined as well as V. In this section, we shall see that the irreversibility of vacuuming poses certain constraints on which modifications are allowed. For example, it makes no sense to insert a specification part in order to keep tuples that are already selected by an existing removal specification part.

Section 6.2 proceeds to cover another type of constraint that applies only to the vacuuming of relation V itself; a type constraint which is accomplished via vacuuming specification parts of the form "$\omega(V) : Exp$." Specifically, to achieve the functionality described in this paper, it is necessary to retain a complete record of what has been removed from the database by vacuuming, so not all vacuuming specification parts can simply be removed.

Throughout we use the vacuuming relation V in Table 5 for illustration. A summary concludes the section.

6.1 Irreversibility-Induced Constraints on Vacuuming

When updating the vacuuming of regular relations and the vacuuming relation, it is a challenge—the only one for vacuuming regular relations—to contend with the irreversibility of vacuuming. For example, once a tuple has been selected by some removal specification part, keep specification parts that would select the tuple must be disallowed. The principle "once vacuumed, always vacuumed" must be satisfied.

Stated precisely, we require that the vacuuming specification V is *growing*, which is defined as follows.

$$growing(V) \stackrel{def}{\Longleftrightarrow} \forall t\big(\forall R_x \ \big(\forall u \ \big(u \in (\llbracket R_x \rrbracket_t, \emptyset) \wedge u \notin (\llbracket R_x \rrbracket_t, \llbracket V \rrbracket_t) \Rightarrow \forall t' > t \ (u \notin (\llbracket R_x \rrbracket_{t'}, \llbracket V \rrbracket_{t'}))\big)\big)\big)$$

So, a vacuuming specification V is *growing* if and only if all tuples u being removed from relation R_x at some time t will continue to be removed for all times t' after t. Note that $(\llbracket R_x \rrbracket_t, \llbracket V \rrbracket_t)$ denotes the relation R_x as it was at time t, vacuumed by the vacuuming specification V as it also was at time t. To ensure V to be growing, we consider (logical) deletions and insertions on V in turn.

There are no restrictions on deletions of keep specification parts. Deletion of a keep specification part cannot result in less being removed, but may result in more being removed, and so does not conflict with the above requirement.

Consider deletion of a general removal specification part v, a tuple of the form ("$\rho(R_x) : \sigma_P(R_x)$", t_{ins}, NOW). This part was inserted at time t_{ins}, remains current, and is thus a candidate for deletion. The

expression for the corresponding vacuuming-modified relation (at time t') is defined as

$$\sigma_{\neg[\exists t\ (P' \wedge t_{ins} \leq t \leq t')]}(R_x), \tag{4}$$

where P' is P with occurrences of NOW replaced by t. The value t' in this expression was obtained by evaluating $[\![TT^{\dashv}]\!]_{t'} = [\![NOW]\!]_{t'}$. Deleting v is done by updating the $TT^{\dashv}$ value of v to t_{del}, the current time when the deletion occurs. The expression for the vacuuming-modified relation (at any time) then becomes

$$\sigma_{\neg[\exists t\ (P' \wedge t_{ins} \leq t \leq t_{del})]}(R_x). \tag{5}$$

To see that the deletion of v does not render a growing specification non-growing, it is sufficient to observe that Expression 5 is at least as restrictive as, i.e., returns no more tuples that, the one it replaces at the time of the deletion, the time Expression 5 replaces Expression 4. At this time, Expression 4 has $t' = t_{del}$, making the two expressions equivalent.

Turning to insertions, first observe that any specification part by itself is growing. Inspecting Expression 4, it can be seen that the range of possible values of variable t increases as time passes. By virtue of the negation, the selection predicate thus becomes more and more restrictive as time passes. The expression for a deleted part, given in Expression 5, remains constant. These observations also hold for keep specification parts.

One problem remains: It is possible for a keep specification part to select a tuple already selected by a removal specification part, creating an impossible situation where a tuple selected for removal and possibly already removed must be kept in the database. This situation may occur because of the insertion of either a removal or a keep specification part. Before stating requirements to insertions that avoid this problem, we give examples that explore the issues involved.

	Vspec		$TT^{\vdash}$	$TT^{\dashv}$
v_1	$\rho(emp)$	$: \sigma_{TT^{\dashv} \leq NOW-4yrs}(emp)$	$5/16/1992$	$7/14/1997$
v_2	$\kappa(emp)$	$: \sigma_{Bal \leq \$-5{,}000}(emp)$	$5/16/1992$	NOW
v_3	$\rho(emp)$	$: \sigma_{NOW-4yrs < TT^{\dashv} \leq NOW-2yrs \wedge Sex=F}(emp)$	$7/4/1996$	NOW
v_4	$\rho(emp)$	$: \sigma_{TT^{\dashv} \leq NOW-6yrs}(emp)$	$7/15/1997$	NOW
v_5	$\kappa(emp)$	$: \sigma_{7/15/3996-NOW \leq TT^{\dashv} \leq 7/15/4000-NOW}(emp)$	$7/14/1998$	NOW
v_6	$\rho(V)$	$: \sigma_{TT^{\dashv} \leq NOW}(V)$	$7/14/1998$	NOW
v_7	$\rho(emp)$	$: \sigma_{TT^{\dashv} \leq NOW-1yrs \wedge Sex=M}(emp)$	$7/14/1998$	NOW
v_8	$\kappa(emp)$	$: \sigma_{TT^{\dashv} \geq NOW-3yrs}(emp)$	$7/14/1998$	NOW

Table 5: Vacuuming Specification V at Time 7/14/1998

EXAMPLE: Table 5 gives the vacuuming specification V, with the current time being 7/14/98. The specification parts are ordered and numbered according to their insertion time. Specification parts v_5 to v_8 are to be inserted at the current time.

Inserting a removal specification part may lead to a conflict with an existing keep specification part, as we show next. When this occurs, the removal specification part should not be inserted. Now assume that V consists of only the keep specification part v_5, which has just been inserted. At time t', v_5 states that tuples of *emp* satisfying the following predicate must be retained.

$$\exists t\ (7/15/3996 - t \leq TT^{\dashv} \leq 7/15/4000 - t\ \wedge\ 7/14/1998 \leq t \leq t')$$

For example, for t' = 7/14/1998, the predicate is "$1/1/1998 \leq TT^{\dashv} \leq 1/1/2002$," and the lower bound on $TT^{\dashv}$ will continue to decrease as time passes.

Now assume that we want to insert specification part v_7 from Table 5. The semantics of v_7 at time t' is

$$\sigma_{\neg\,[\exists t\ (TT^{\dashv} \leq t'-1yrs\ \wedge\ Sex=M\ \wedge\ 7/14/1998 \leq t \leq t')]}(emp).$$

For t' = 7/14/1998, the selection predicate becomes "$\neg[TT^{\dashv} \leq$ 7/14/1997 $\wedge\, Sex = M]$," and the upper limit on $TT^{\dashv}$ will increase as the current time increases.

Inserting v_7 will not present a problem at the time of insertion, but after a few months, a situation will occur where what v_5 says must be kept has already been selected for removal by v_7. For example, in six months the lower bound on $TT^{\dashv}$ in the expression for v_5 is 7/1/1997. But v_5 selects tuples with $TT^{\dashv} \leq$ 7/14/1997 (and with $Sex = M$) for removal already at the current time. In conclusion, insertion of v_7 is not acceptable.

Note that, going back in time to a prior situation, having $V = \{v_1, v_2\}$ at current time 7/4/1996 and inserting v_3 or v_4 at some later time does not present problems. v_2 is the only keep specification part, and it does not expand to select tuples that are selected by the removal specification parts as time increases. □

Requirering an insertion of a removal specification part to be *growth assuring* generalizes the observations in the example. Assume that removal specification part v_i concerns relation R_x. When tried for insertion into specification V at time t, insertion of v_i is growth assuring if $growRem(v_i, t, V)$, defined as follows.

$$\begin{aligned} growRem(v_i, t, V) \ &\stackrel{def}{\Longleftrightarrow} \\ &\neg\,[\exists t', t''\ (t \leq t' < t'' \wedge \exists u\ (u \notin ([\![R_x]\!]_{t'}, [\![V \cup \{v_i\}]\!]_{t'}) \wedge u \in ([\![R_x]\!]_{t''}, [\![V \cup \{v_i\}]\!]_{t''})))] \end{aligned}$$

The definition states that insertion of a removal specification part is growth assuring if no two times t' and t'' later than the insertion time exist so that a tuple u can be found not being in the vacuumed relation at time t', but being in the vacuumed relation at the later time t''.

Turning to the insertion of keep specification parts, two similar problems can occur. A keep specification part to be inserted can specify that tuples already selected for removal must be kept, or the keep specification part can at some future time select tuples for keeping that were selected for removal prior to that time. The next example illustrates this.

EXAMPLE: Assume that $V = \{v_1, v_2, v_3, v_4\}$ and that we want to insert v_8 (see Table 5).

At the current time, 7/14/1998, specification v_3 selects tuples satisfying predicate $8/30/1991 \leq TT^{\dashv} \leq 7/14/1996 \wedge Sex = F$ for removal. Since v_8 currently specifies that tuples satisfying predicate $TT^{\dashv} \geq 7/14/1995$ should be kept, inserting v_8 will create an instant problem and cannot be inserted.

Inserting v_5 creates a delayed problem. For example, after the date 1/1/2000, v_5 will specify that tuples should be kept if (logically) deleted on or after 7/14/1996, but v_3 already selects tuples deleted at that date for removal. So in time, also inserting v_5 will create a problem. □

Assume that keep specification part v_j concerns relation R_x. Then, when tried for insertion into specification V at time t, insertion of v_j is *growth assuring* if $growKeep(v_j, t, V)$, defined as follows.

$$\begin{aligned} growKeep(v_j, t, V) \ &\stackrel{def}{\Longleftrightarrow} \\ &\neg\,[\exists t'\ (t' < t \wedge \exists u'\ (u' \notin ([\![R_x]\!]_{t'}, [\![V]\!]_{t'}) \wedge u' \in ([\![R_x]\!]_{t}, [\![V \cup \{v_j\}]\!]_{t})))] \wedge \\ &\neg\,[\exists t', t''\ (t \leq t' < t'' \wedge \exists u\ (u \notin ([\![R_x]\!]_{t'}, [\![V \cup \{v_j\}]\!]_{t'}) \wedge u \in ([\![R_x]\!]_{t''}, [\![V \cup \{v_j\}]\!]_{t''})))] \end{aligned}$$

The first line in the definition states that, at the time of insertion, no tuple u' must exist that is selected for removal by V before that time, but not by the modified specification. The second line has the same format as the definition of growth assuring for insertions of removal specification parts.

6.2 Retention of Vacuuming Information

The vacuuming specification is itself a temporal relation, and so it is possible to also apply vacuuming to the vacuuming specification itself. However, we must ensure that a complete record is retained of the vacuuming that is or will be in effect. This section formulates constraints that ensure this.

It should be clear that removal of specification parts being current is problematic. Even parts that have been deleted may not always be selected for removal. An example illustrates the potential problem.

EXAMPLE: Consider specification parts v_1 and v_4 in Table 5. Here v_4 takes the place of v_1 at time 7/15/1997. Now, at the current time 7/14/1998, even though v_1 is deleted, it is still the reason for the removal of tuples with $TT^{\dashv} \leq 7/14/1993$, and v_4 still has no tuples to remove, since it selects tuples deleted before 7/14/1992 for removal. Thus, v_1 although not current is still important if one is to understand the contents of the database.

Consider also v_6 specifying the removal of vacuuming specification parts that have been deleted. This specification would remove specification part v_1. If that happens, it will, for some time, not be possible to see that original data may have been removed. Due to this, specification part v_6 should not be allowed. □

To ensure that relevant information about vacuuming is not lost, we introduce the notions of *active* and *passive* specification parts. A specification part v of a vacuuming specification V is *active* at time t if it is responsible for vacuuming at time t. We define this for a specification part v specifying vacuuming for relation R_x.

$$\begin{aligned} active(v,t,V) \quad &\stackrel{def}{\iff} \\ &[\exists u\ (u \notin (\llbracket R_x \rrbracket_t, \llbracket V \rrbracket_t) \wedge u \in (\llbracket R_x \rrbracket_t, \llbracket V \setminus \{v\} \rrbracket_t)) \wedge \exists Exp\ (v.Vspec = \rho(R_x) : Exp)] \vee \\ &[\exists u\ (u \in (\llbracket R_x \rrbracket_t, \llbracket V \rrbracket_t) \wedge u \notin (\llbracket R_x \rrbracket_t, \llbracket V \setminus \{v\} \rrbracket_t)) \wedge \exists Exp\ (v.Vspec = \kappa(R_x) : Exp)] \end{aligned}$$

A removal specification part v_i specifying vacuuming for R_x is active at time t if a tuple u exists, with u being in relation $\llbracket R_x \rrbracket_t$ vacuumed by V excluding v_i, and with u not being in $\llbracket R_x \rrbracket_t$ vacuumed by all of V. In the same way, a keep specification part v_j is active at time t if a tuple u exists, with u being in $\llbracket R_x \rrbracket_t$ vacuumed by all of V, but not being in $\llbracket R_x \rrbracket_t$ being vacuumed by V excluding v_j. Removal and keep specification parts are thus active if their presence select additional tuples for removal and keep, respectively.

Next, a specification part, removal or keep, is passive if it is not active, but will be so at a later time.

$$passive(v,t,V) \quad \stackrel{def}{\iff} \quad \neg\ active(v,t,V)\ \wedge\ \exists t' > t\ (active(v,t',V))$$

The *active* and *passive* specification parts may at some time be responsible for vacuuming. This makes these parts *alive* in contrast to the ones that will never be responsible for vacuuming—the *dead* ones, see Figure 2. For all specification parts $v \in V$ and time t we define these predicates next.

$$\begin{aligned} alive(v,t,V) \quad &\stackrel{def}{\iff} \quad active(v,t,V) \vee passive(v,t,V) \\ dead(v,t,V) \quad &\stackrel{def}{\iff} \quad \neg\ alive(v,t,V) \end{aligned}$$

Finally, the sets of active and alive specification parts at time t may be defined as follows.

$$\begin{aligned} active\,(V,t) &= \{v \mid v \in \llbracket V \rrbracket_t \wedge active(v,t,V)\} \\ alive\,(V,t) &= \{v \mid v \in \llbracket V \rrbracket_t \wedge alive(v,t,V)\} \end{aligned}$$

The first set is exactly the specification parts responsible for the vacuuming at the time t, and the second set is the parts that either are responsible for vacuuming at time t or will be so at a later time.

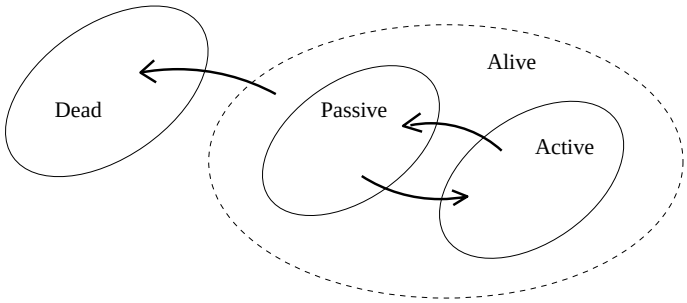

Figure 2: Migration of Types of Vacuuming Specification Parts

EXAMPLE: To illustrate, let $V = \{v_1, v_2, v_3, v_4\}$ be the current vacuuming specification at time 7/14/1998 as given in Table 5. Then $active(V, 7/14/1998) = \{v_1, v_2, v_3\}$. At this time, part v_4 selects only tuples deleted before 7/14/1992 for removal, but v_1 also selects these and more tuples for removal. So at time 7/14/1998, v_4 is not active. But v_4 is passive and thus alive because it will be active after time 7/14/1999. Note also that when v_4 becomes *active* after time 7/14/1999, v_1 will be *dead*. □

To ensure that no relevant vacuuming information is lost, the system must retain all specification parts in the set of alive parts. So, when modifying the vacuuming relation at some time t, all that is necessary is to check if parts that are in $alive(V, t)$ will be removed. Note that the vacuuming specification must continue to be *growing*.

Now, modifying vacuuming on the vacuuming relation corresponds to deleting and inserting tuples in $V|_V$.

Deleting tuples will not create any problems. Deleting a vacuuming specification part results in a fixed timestamp end value in the tuple. This only stops the vacuuming, retaining existing vacuuming knowledge. Thus, no vacuuming knowledge is lost and, for the same reason as above, the vacuuming specification will continue to be *growing*.

However, inserting tuples can create problems, since this will specify removal or keep of vacuuming specification parts. First of all, it is still a possibility that the part to be inserted will make the vacuuming specification non-growing. This was addressed in the previous subsection. Second, inserting a keep specification part will not create further problems, but inserting removal specification parts for V will create a potential loss of vacuuming knowledge. To ensure that this will not happen, specifying removal of specification parts being *alive* should not be allowed.

So, what makes a removal specification part v_i, specifying vacuuming on V, admissible for insertion into V? First, as stated before, the insertion must assure growth, and second it must be *information retaining*. Inserting v_i into V retains vacuuming information if $infRet(v_i, t, V)$, defined as follows.

$$infRet(v_i, t, V) \stackrel{def}{\Longleftrightarrow} \neg\,[\exists t'\ (t' \geq t \wedge \exists v'\ (alive(v', t', [\![V]\!]_{t'}) \wedge v' \in ([\![V]\!]_{t'}, [\![V]\!]_{t'}) \wedge v' \notin ([\![V]\!]_{t'}, [\![V \cup \{v_i\}]\!]_{t'})))]$$

The definition says insertion of a removal specification part v_i at the time t retains information about specification parts being alive, if and only if there at no time t' after t exists a vacuuming specification part being alive at t', and being removed by v_i at that time, i.e., the insertion retains information if only dead parts will be removed by v_i.

6.3 Summary

So, two major problems can occur when inserting new specification parts. First, the insertion can violate the principle "once vacuumed, always vacuumed," and second the insertion can create a loss of vacuuming knowledge. To secure that none of these problems occur, we have defined the properties for these cases in the prior subsections. The full definition of admissibility for insertions is given next.

$$
\begin{aligned}
&admInsertion(v, t, V) \stackrel{def}{\Longleftrightarrow} \\
&\quad [infRet(v, t, V) \wedge growRem(v, t, V) \wedge \exists Exp\ (v.Vspec = \rho(V) : Exp)]\ \vee \\
&\quad [growRem(v, t, V) \wedge \exists Exp, R_x\ (v.Vspec = \rho(R_x) : Exp))]\ \vee \\
&\quad [growKeep(v, t, V) \wedge \exists Exp, R_x\ (v.Vspec = \kappa(R_x) : Exp \vee v.Vspec = \kappa(V) : Exp))]
\end{aligned}
$$

7 Conclusions and Research Directions

A wide range of applications are faced with accountability and trace-ability requirements, in turn yielding underlying databases that retain their past states. Such databases, termed transaction-time databases, are ever growing, and even logical deletions result in insertions at the physical level.

This paper presents a semantic framework for the physical removal of data, or vacuuming, from such databases. While necessary, vacuuming compromises the property that past database states are retained. The framework defines the semantics of vacuuming specification facilities, and it supports the detection of queries that, if answered, may yield results affected by vacuuming. This support provides the foundation for offering user-friendly query support on vacuumed databases, which is also covered. The detection of vacuuming-affected queries imposes certain constraints on the modification of the vacuuming specifications that are in effect; the concepts necessary to capture these constraints as well as the constraints themselves are given.

The studies reported in this paper point to interesting research directions, some of which are described next.

In the current framework, vacuuming is an "all-or-nothing" proposition: either data is irreversibly eliminated or is retained. Extending the framework to also allow for the specification of off-line (or even "near-line") archival in the context of multi-level storage architectures appears to be an interesting and very useful, but also non-trivial direction.

One of today's foci in data warehousing is the bulk-loading of very large amounts of data, but as years of data are accumulating in data warehouses, vacuuming is likely to become a future focus of attention. The advanced decision support queries in data warehousing are expected to introduce new challenges.

When a query against a vacuumed database may not return the same result as when issued against the unvacuumed, but otherwise identical database, a cooperative system may offer alternative queries that are similar to the original query and that are unaffected by vacuuming. The use of techniques such as query generalization and specialization for obtaining simple and easily comprehensible alternative queries deserves exploration.

Acknowledgements

This research was supported in part by the Danish Research Councils through grants 9700780 and 9701406, by the CHOROCHRONOS project, funded by the European Commission, contract no. FMRX-CT96-0056, and by a grant from the Nykredit corporation.

References

[ASU79a] A. V. Aho, Y. Sagiv, and J. D. Ullman. Efficient Optimization of a Class of Relational Expressions. *ACM Transactions on Database Systems*, 4(4):435–454, December 1979.

[ASU79b] A. V. Aho, Y. Sagiv, and J. D. Ullman. Equivalences Among Relational Expressions. *SIAM Journal of Computing*, 8(2):218–246, May 1979.

[CDI+97] J. Clifford, C. Dyreson, T. Isakowitz, C. S. Jensen, and R. T. Snodgrass. On the Semantics of "Now" in Databases. *ACM Transactions on Database Systems*, 22(2):171–214, June 1997.

[Cha90] S. Chaudhuri. Generalization as a Framework for Query Modification. In *Proceedings of the 6th Data Engineering Conference*, pages 138–145, February 1990.

[Cop82] G. Copeland. What If Mass Storage Were Free? *IEEE Computer Magazine*, 15(7):27–35, July 1982.

[GMLY98] H. Garcia-Molina, W. Labio, and J. Yang. Expiring Data in a Warehouse. To appear in *Proceedings of the 24th International Conference on Very Large Databases*, August 1998.

[Jen95] C. S. Jensen. Vacuuming. In R. T. Snodgrass, editor, *The TSQL2 Temporal Query Language*, Chapter 23, pages 451–462. Kluwer Academic Publishers, 1995.

[JM90] C. S. Jensen and L. Mark. A Framework for Vacuuming Temporal Databases. Technical report, CS-TR-2516, UMIACS-TR-90-105, Department of Computer Science. University of Maryland, College Park, MD 20742, August 1990.

[Mot84] A. Motro. Query Generalization: A Technique for Handling Query Failure. In *Proceedings of the 1st International Workshop on Expert Database Systems*, pages 314–325, October 1984.

[MS91] E. McKenzie and R. Snodgrass. Evaluation of Relational Algebras Incorporating the Time Dimension in Databases. *Computing Surveys*, 23(4):501–543, 1991.

[PS88] J. Park and A. Segev. Using Common Subexpressions to Optimize Multiple Queries. In *Proceedings ofthe 4th Data Engineering Conference*, pages 311–319, February 1988.

[RS87] L. A. Rowe and M. R. Stonebraker. The Postgres Papers. Memorandum UCB/ERL M86/85, Electronics Research Laboratory, College of Engineering, University of California, Berkeley, CA 94720, June 1987.

[SA85] R. T. Snodgrass and I. Ahn. A Taxonomy of Time in Databases. In *Proceedings of ACM SIGMOD*, pages 236–246, May 1985.

[SAA+94] R. T. Snodgrass, I. Ahn, G. Ariav, D. S. Batory, J. Clifford, C. E. Dyreson, R. Elmasri, F. Grandi, C. S. Jensen, W. Kafer, N. Kline, K. Kulkarni, T. Y. C. Leung, N. Lorentzos, J. F. Roddick, A. Segev, M. D. Soo, and S. M. Sripada. TSQL2 Language Specification. *SIGMOD Record*, 1(23):65–86, March 1994.

[SC75] J. M. Smith and P. Yen-Tang Chang. Optimizing the Performance of a Relational Algebra Interface. *Communications of the ACM*, 18(10):569–579, October 1975.

[Sch77] B. M. Schueler. Update Reconsidered. In *Proceedings of the IFIP Working Conference on Modelling in Data Base Management Systems*, pages 149–164, 1977.

[Sto75] M. R. 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.

[Ull88a] J. D. Ullman. *Database and Knowledge—Base Systems*, Volume I of *Principles of Computer Science*. Computer Science Press, Rockville, MD, 1988.

[Ull88b] J. D. Ullman. *Database and Knowledge—Base Systems*, Volume II of *Principles of Computer Science*. Computer Science Press, Rockville, MD, 1988.