# Expiration of Historical Databases

(Extended Abstract)

David Toman

Department of Computer Science, University of Waterloo
Waterloo, Ontario, Canada N2L 3G1
E-mail: david@uwaterloo.ca

## Abstract

*We present a technique for automatic expiration of data in a historical data warehouse that preserves answers to a known and fixed set of first-order queries. In addition, we show that for queries with output size bounded by a function of the active data domain size (the number of values that have ever appeared in the warehouse), the size of the portion of the data warehouse history needed to answer the queries is also bounded by a function of the active data domain size and therefore does not depend on the age of the warehouse (the length of the history).*

## 1 Introduction

Data warehousing applications often require the data warehouse to store *histories* of the evolution of a data warehouse rather than just the *current (latest) state* to allow subsequent querying of the warehouse histories, e.g., for the purposes of data analysis. With the progression of time, however, the size of such histories grows, requiring an ever increasing amount of storage.

Approaches based on lossless storage of the data warehouse histories, e.g., those using compression techniques, are unlikely to be able to handle the growth of the warehouse indefinitely [2]. Even if adding storage to accommodate the growth of the data size were possible, the growing amount of data needs to be examined during *querying* and leads to deterioration of query performance over time.

Other, often more practically oriented approaches, resort to truncation of the history beyond a given point in time based on various ad-hoc *policies*, e.g., tax records have to be kept for 5 years. However, these approaches necessarily lead to information loss that may adversely affect the data warehousing queries. In general there may not be a point in the history such that all data older than this point can be safely deleted while preserving query answers.

This paper studies the possibility of physically removing redundant data from the history and the *limits* of such approaches. The main contribution of the paper is a technique that allows automatic expiration of a warehouse history while preserving the capability to answer a fixed set of *arbitrary* first-order queries. The technique guarantees that the expired history's size is always bounded by a function of the number of data constants in the history and the approximated size of the query answer. In particular, for large classes of first-order queries the size of the expired history is *independent* of the length of the history. In addition, the proposed technique has the following features:

- It always outperforms the naive storage of the whole history with only a tiny penalty due to auxiliary data.
- It is asymptotically optimal for techniques based on removing states in the history. We provide an example query that matches the upper space bound for the technique. Other ways of encoding histories, however, may use less space in general.
- It can be implemented using first-order updates on top of a standard SQL engine.
- The proposed expiration technique does not have to be executed after every update of the data warehouse/clock tick. Instead it can be run *on demand*, e.g., when the history grows beyond a given limit. This feature is significant if the method is used to implement a background *garbage collector* for warehouse histories.

### 1.1 Related Work

General techniques for expiration of data in historical data warehouses can be divided into three broad categories:

**Methods based in View (Self-)Maintenance.** These approaches take advantage of algorithms for materialized view self-maintenance [7, 10, 14]; Gupta and Mumick [6] give an overview of various view maintenance approaches and

Colby et al. [4] gives a compact and precise description of maintenance algorithms for first-order queries.

Yang and Widom [15, 16] applied this approach to the maintenance of temporal views over evolving data sources. Their query/view definition language is based on the temporal relational algebra that is used in TSQL2 [11] for snapshot-reducible queries. However, their results were already subsumed by Chomicki's work on temporal integrity constraints formulated in past temporal logic [2] since temporal relational algebra has been shown to be equivalent to temporal logic [1, 13]. In addition, Chomicki provides a space bound for his method: the size of auxiliary materialized views is bounded by a polynomial in the size of the active data domain of the history and *is independent* of the length of the history.

In general, approaches based on view self-maintenance are limited by the classes of queries that can be self-maintained. Also, it has been shown, that in general, the auxiliary information needed by many of the self-maintenance algorithms may take as much as polynomially[1] more space than the original history [2].

**Methods based on Physical Deletion.** Another approach is based on removing past states that do not contribute to answering the given query. These techniques have been studied by Garcia-Molina et al. [5]. That paper however, does not provide a bound on the size of the expired history; indeed their techniques are based on safe expiration of "old" data first; it is easy to see that such an approach may fail to reduce of the size of the data warehouse if we insist on preserving answers of fixed set of queries:

**Example 1.1** Consider an instance of a historical warehouse with schema containing a single unary relation $R$ that always contains a single value at every point in time. At some time $n > 0$ the history may look as follows:

$$\{a\}_0, \{b\}_1, \{b\}_2, \ldots, \{b\}_n$$

Then to be able to correctly answer a query $\{x : \exists t.R(t, x)\}$ it is easy to see that we cannot remove any *prefix* of the history. □

On the other hand, Garcia-Molina et al. [5] survey many possible topics of interest and how they relate to data expiration, including the impact of integrity constraints, incomplete information, and updates.

**Methods based on Query Specialization (Partial Evaluation [8]).** Yet another approach can be based on *specialization* of queries: the expiration algorithm transforms a given query with respect to the known part of the history: it treats the known history as a simple *constraint* formula [9] that, combined with the original query, yields a *residual query* that only depends on the *future extensions* of the current history. The residual formula is then simplified by removing empty/redundant subformulas that can be detected syntactically (undecidable in general).

The method proposed in this paper overlaps the later two categories. It provides, however, a general solution for the problem in the above example. It does so for any finite set of range-restricted first-order queries, a language strictly more expressive than temporal relational algebra [1, 13], and therefore more expressive than the languages used by Yang and Widom [15, 16]. Also, unlike most other approaches, save Chomicki's work on temporal integrity constraints [2], the technique comes with a tight space bound on the size of the expired history. In particular, the size of the expired history is independent of the length of the original history for a large class of first-order queries. In addition, and again unlike the approaches based on view maintenance, the size of the expired history is essentially bounded by the size of the original history.

### 1.2 Overview of the paper

The rest of the paper is organized as follows: Section 2 introduces data-warehouse histories and queries. Section 3 presents a general expiration framework and defines the notion of bounded encoding of histories. Section 4 introduces the actual expiration method. Section 5 briefly comments on implementation and possible optimizations. We conclude the paper by listing several possibilities of future extensions of the proposed approach, in particular, we discuss the problems caused by general aggregation (counting) operators.

## 2 Historical Data Warehouses

In this paper we assume that time is modeled by positive integers; we discuss other possibilities, e.g., the use of interval-based encodings in Section 6. We model the individual states of the data warehouses as relational databases. In this setting a finite history of a data warehouse is simply a time (integer) indexed sequence of relational databases:

**Definition 2.1 (History)** Let $\rho$ be a relational signature. A *historical data warehouse* $D$ (or a *history* for short) is an finite integer-indexed sequence of databases

$$D = (D_0, D_1, \ldots, D_n)$$

where $D_i$ is a standard relational database over $\rho$. We call $D_i$ a *state* of $D$ and $i$ a *time instant*.

[1] With the degree of the polynomial bounded by the size of the maintained query.

The *data domain* $\mathbf{dom}_D$ of a history $D$ is the union of all data values that appear in any relation in $D_i$ at any time instant; the *temporal domain* $\mathbf{dom}_T$ is the set of all time instants that appear as indices in the history $D$.

For a history $D$ we define $\mathsf{Max}(D)$ to be the maximal (latest) time instant in $\mathbf{dom}_T$. □

A history $D$ can be *extended* by adding a database $D_j$, $j > \mathsf{Max}(D)$ to the end of the sequence. This process can be repeated arbitrarily many times. Let $D'$ be a sequence of all states successively added to $D$. We call $D'$ a *suffix* of $D$ and write $D; D'$ for the extension of $D$ by $D'$.

We use the standard syntax for range-restricted first-order queries to query the histories of the data warehouse.

**Definition 2.2 (Queries)** We use the following BNF to specify first-order queries:

$$Q ::= R(t, \mathbf{x}) \mid \exists x.Q \mid \exists t.Q \mid Q \wedge Q \mid Q \wedge F \mid Q \wedge \neg Q \mid Q \vee Q$$

where $R$ is a relational symbol, $\mathbf{x}$ is a tuple of variables, and $F$ is of the form $x = y$ for data variables and $t = s$ or $t < s$ for temporal variables. We require the queries to obey the standard syntactic safety rules: variables in a condition $F$ must appear free in the accompanying query and free variables of subqueries involved in disjunction or negation must match. We also assume that the quantified variables have unique names different from all other variables in the query[2].

The semantics of the queries is defined using the usual satisfaction relation $\models$ that links histories ($D$) and substitutions ($\theta$) with queries; the only difference is in the case of base relations: for $R(\mathbf{x}) \in \rho$ we define $D, \theta \models R(t, \mathbf{x})$ if $\mathbf{x}\theta \in R(D_{t\theta})$. In other words, the base relations are evaluated at the point of the warehouse history specified by their first argument. We assume that the valuations $\theta$ always map variables to values of the appropriate domain and are restricted to the free variables of the particular query. □

## 3 Expiration of Histories

Various approaches to expiration[3] of histories can be essentially characterized by an *expiration operator* $\mathcal{E} : Q, D \to Q^{\mathcal{E}}, D^{\mathcal{E}}$ that maps histories and queries to their *expired* versions. The operator must satisfy the following requirements:

$$\begin{aligned} Q(D) &= Q^{\mathcal{E}}(D^{\mathcal{E}}) && \text{(answer preservation)} \\ \mathcal{E}(Q, D; D') &= \mathcal{E}(Q, D^{\mathcal{E}}; D') && \text{(extension maintenance)} \end{aligned}$$

The methods differ in the space needed to store the residual database and query, $|Q^{\mathcal{E}}| + |D^{\mathcal{E}}|$. This characteristic can be measured with respect to several parameters, in particular the sizes of the active data and temporal domains ($\mathbf{dom}_D$ and $\mathbf{dom}_T$) of $D$ and the size of $Q$. An expiration method provides a *bounded encoding of a history* if $|Q^{\mathcal{E}}| + |D^{\mathcal{E}}|$ do not depend on the length of the history [2].

**Example 3.1** It is easy to see that allowing arbitrary first order queries forces us immediately to keep the whole history of the data warehouse:

(1) Consider a query $R(t, \mathbf{x})$ for $R \in \rho$. The correct answer consist of the whole history of $R$ in the warehouse; therefore we need to know (and thus store) the whole history to compute the answer for the above query.

(2) Compare this situation to having just a single query $\exists t.R(t, \mathbf{x})$. In this case we can simply remember a single time instant for every valuation of $\mathbf{x}$ that ever appeared in the data warehouse and eliminate the rest. □

In the light of the above example we need to restrict our attention to queries with *bounded answers*: $|Q(D)| \leq f(|\mathbf{dom}_D|)$. Unfortunately:

**Theorem 3.2** *Whether a query is semantically bounded is undecidable.* □

In the rest of the paper we therefore restrict our attention to queries that are *syntactically* bounded: we require all free variables in the query to be *data* variables. This class contains an important subclass of *first-order temporal integrity constraints* (closed queries). Therefore the method proposed in this paper handles a strictly larger class of queries compared to the method proposed by Chomicki for Past Temporal Logic-based integrity constraints [2]. The strictness follows from results on separation of temporal logic from first-order two-sorted logic [1, 13].

## 4 Query Specialization-based Expiration

We now turn our attention to the main goal of the paper: to finding out which values (states) in a data warehouse are not needed to evaluate a given query (or a set of queries) in any future extension of the current history of the warehouse. There are two principal situations in which a value is no longer needed, intuitively:

1. the value cannot match any selection condition in the given query *in any possible extension of the current history*. Thus this value can not *contribute* to answering of the query and can be removed.
2. the value contributes to answering the query but there is another value that provides *the same answer*; thus one of these values is redundant and can be removed as well.

[2] We can add constants in the $F$ formulas without affecting the result.

[3] We use the word *expiration* in a broader sense to describe methods that encode and reduce the size of database histories.

$$
\mathsf{PE}_D(Q) = \begin{cases}
\{\mathrm{true}\genfrac{[}{]}{0pt}{3}{t\mathbf{x}}{s\mathbf{a}} : R(s,\mathbf{a}) \in D\} & \\
\qquad \cup \{R(t,\mathbf{x})\genfrac{[}{]}{0pt}{3}{t\mathbf{x}}{\bullet\mathbf{a}} : \mathbf{a} \in (\mathbf{dom}_D \cup \{\bullet\})^{|\mathbf{x}|}\} & Q \equiv R(t,\mathbf{x}) \\
\{Q_1'\genfrac{[}{]}{0pt}{3}{\mathbf{x}}{\mathbf{a}} : Q_1'\genfrac{[}{]}{0pt}{3}{\mathbf{x}}{\mathbf{a}} \in \mathsf{PE}_D(Q_1), \models \genfrac{[}{]}{0pt}{3}{\mathbf{x}}{\mathbf{a}} \wedge F\} & Q \equiv Q_1 \wedge F \\
\{Q_1' \wedge Q_2'\genfrac{[}{]}{0pt}{3}{\mathbf{xy}}{\mathbf{ab}} : Q_1'\genfrac{[}{]}{0pt}{3}{\mathbf{x}}{\mathbf{a}} \in \mathsf{PE}_D(Q_1), Q_2'\genfrac{[}{]}{0pt}{3}{\mathbf{y}}{\mathbf{b}} \in \mathsf{PE}_D(Q_2), \models \genfrac{[}{]}{0pt}{3}{\mathbf{xy}}{\mathbf{ab}}\} & Q \equiv Q_1 \wedge Q_2 \\
\{(\exists y. \bigvee_{Q_1'\genfrac{[}{]}{0pt}{3}{\mathbf{x}y}{\mathbf{a}b} \in \mathsf{PE}_D(Q_1)} Q_1')\genfrac{[}{]}{0pt}{3}{\mathbf{x}}{\mathbf{a}} : \exists b. Q_1''\genfrac{[}{]}{0pt}{3}{\mathbf{x}y}{\mathbf{a}b} \in \mathsf{PE}_D(Q_1)\} & Q \equiv \exists y.Q_1 \\
\{(\exists t. \bigvee_{Q_1'\genfrac{[}{]}{0pt}{3}{\mathbf{x}t}{\mathbf{a}s} \in \mathsf{PE}_D(Q_1), s \in \mathsf{TB}_{\mathbf{a}}(t)} Q_1')\genfrac{[}{]}{0pt}{3}{\mathbf{x}}{\mathbf{a}} : \exists s. Q_1''\genfrac{[}{]}{0pt}{3}{\mathbf{x}t}{\mathbf{a}s} \in \mathsf{PE}_D(Q_1)\} & Q \equiv \exists t.Q_1 \\
\{Q_1' \wedge \neg Q_2'\genfrac{[}{]}{0pt}{3}{\mathbf{x}}{\mathbf{a}} : Q_1'\genfrac{[}{]}{0pt}{3}{\mathbf{x}}{\mathbf{a}} \in \mathsf{PE}_D(Q_1), Q_2'\genfrac{[}{]}{0pt}{3}{\mathbf{x}}{\mathbf{a}} \in \mathsf{PE}_D(Q_2)\} & \\
\qquad \cup \{Q_1'\genfrac{[}{]}{0pt}{3}{\mathbf{x}}{\mathbf{a}} : Q_1'\genfrac{[}{]}{0pt}{3}{\mathbf{x}}{\mathbf{a}} \in \mathsf{PE}_D(Q_1), Q_2'\genfrac{[}{]}{0pt}{3}{\mathbf{x}}{\mathbf{a}} \notin \mathsf{PE}_D(Q_2)\} & Q \equiv Q_1 \wedge \neg Q_2 \\
\{Q_1' \vee Q_2'\genfrac{[}{]}{0pt}{3}{\mathbf{x}}{\mathbf{a}} : Q_1' \in \mathsf{PE}_D(Q_1)\genfrac{[}{]}{0pt}{3}{\mathbf{x}}{\mathbf{a}}, Q_2'\genfrac{[}{]}{0pt}{3}{\mathbf{x}}{\mathbf{a}} \in \mathsf{PE}_D(Q_2)\} & \\
\qquad \cup \{Q_1'\genfrac{[}{]}{0pt}{3}{\mathbf{x}}{\mathbf{a}} : Q_1'\genfrac{[}{]}{0pt}{3}{\mathbf{x}}{\mathbf{a}} \in \mathsf{PE}_D(Q_1), Q_2'\genfrac{[}{]}{0pt}{3}{\mathbf{x}}{\mathbf{a}} \notin \mathsf{PE}_D(Q_2)\} & \\
\qquad \cup \{Q_2'\genfrac{[}{]}{0pt}{3}{\mathbf{x}}{\mathbf{a}} : Q_1'\genfrac{[}{]}{0pt}{3}{\mathbf{x}}{\mathbf{a}} \notin \mathsf{PE}_D(Q_1), Q_2'\genfrac{[}{]}{0pt}{3}{\mathbf{x}}{\mathbf{a}} \in \mathsf{PE}_D(Q_2)\} & Q \equiv Q_1 \vee Q_2
\end{cases}
$$

**Figure 1. Query Specialization and Reduction.**

An important observation at this point is that (1) alone can only be *approximated* by a computable expiration technique. This follows from undecidability of query emptiness for first-order queries: for an unsatisfiable (closed) query $Q$ over $\rho$, the appropriate expiration policy is "remove everything" as the answer will always be empty, and vice versa. On the other hand, to achieve a *bounded encoding*, the approximation used has to be tight enough: while most of the current techniques concentrate on (1) we will see that (2) is the crux to obtaining a bounded encoding.

The proposed solution is based on two techniques: first we *specialize* the given query with respect to the known part of the history to detect values that can be removed (section 4.1), and second, we extract a *residual history* from the specialized query (section 4.2). We show that the obtained history is equivalent to the original history with respect to the query and that the size of the new history is bounded by the size of the active data domain.

## 4.1 Specialization of Queries

The partial-evaluation technique is based on treating relations in the known history $D$ and in all its possible extensions $D'$ as *characteristic* formulas based on equality constraints as follows:

**Definition 4.1 (Abstract Substitution)** Let $D$ be a history and $x$ and $t$ a data and a temporal variables, respectively, and $\bullet \notin \mathbf{dom}_D \cup \mathbf{dom}_T$ a new symbol; this symbol is used to denote all the values outside of the (current) active data and temporal domains. We define *abstract substitutions* to be the formulas

$$
\genfrac{[}{]}{0pt}{3}{x}{a} \equiv \begin{cases} x = a & a \in \mathbf{dom}_D \\ \forall a \in \mathbf{dom}_D . x \neq a & a = \bullet \end{cases}
$$

for a data variable and

$$
\genfrac{[}{]}{0pt}{3}{t}{s} \equiv \begin{cases} t = s & s \in \mathbf{dom}_T \\ t > \mathsf{Max}(\mathbf{dom}_T) & s = \bullet \end{cases}
$$

for a temporal variable. We allow composite abstract substitutions to denote a (finite) conjunction of the above formulas, e.g., $\genfrac{[}{]}{0pt}{3}{xy}{ab}$ denotes the conjunction of $\genfrac{[}{]}{0pt}{3}{x}{a}$ and $\genfrac{[}{]}{0pt}{3}{y}{b}$. $\Box$

Note that different abstract substitutions always denote disjoint sets. The definition allows us to treat relations as disjunctions of abstract substitutions. In particular, we can substitute these disjunctions for the leaves of the original query and then use the usual simplification rules for first-order formulas to specialize the query:

**Definition 4.2 (Query Specialization)** Let $D$ be a history. We define a function $\mathsf{PE}_D$ that maps a query $Q$ to a set of *residual queries* indexed by *abstract substitutions*, $Q_i\genfrac{[}{]}{0pt}{3}{\mathbf{x}}{\mathbf{a}}$, where $\mathbf{x}$ is the set of free variables of $Q$ and $\mathbf{a}$ is the corresponding set of abstract values (of the appropriate type). The function $\mathsf{PE}_D$ is defined inductively on the structure of $Q$ in Figure 1. $\Box$

**Lemma 4.3** *Let* $Q_1\genfrac{[}{]}{0pt}{3}{\mathbf{x}}{\mathbf{a}}, Q_2\genfrac{[}{]}{0pt}{3}{\mathbf{x}}{\mathbf{a}} \in \mathsf{PE}_D(Q)$. *Then* $Q_1 = Q_2$ *syntactically.* $\Box$

In other words, there is at most one residual formula associated with every abstract substitution since all the cases in Figure 1 are *disjoint*. It is easy to see that the $\mathsf{PE}_D$ operator implicitly removes subformulas (of conjunctions) that are guaranteed not to be satisfiable. However, as we noted at the beginning of this section, this is not sufficient: we also need to eliminate multiple equivalent answers, in particular for the $\exists t.Q'$-like formulas. To achieve this goal we define an equivalence on the residual formulas generated by the $\mathsf{PE}_D$ operator as follows:

$$Q = R\text{:}\ \genfrac{[}{]}{0pt}{}{\mathbf{x}}{\mathbf{a}_1} \sim^D_Q \genfrac{[}{]}{0pt}{}{\mathbf{x}}{\mathbf{a}_2} \iff ((Q_1 = Q_2 = \text{ true}) \vee (\mathbf{a}_1 = \mathbf{a}_2)) \text{ for } Q_1\genfrac{[}{]}{0pt}{}{\mathbf{x}}{\mathbf{a}_1}, Q_2\genfrac{[}{]}{0pt}{}{\mathbf{x}}{\mathbf{a}_2} \in \mathsf{PE}_D\, Q,$$

$$Q = Q' \wedge F\text{:}\ \genfrac{[}{]}{0pt}{}{\mathbf{x}}{\mathbf{a}_1} \sim^D_Q \genfrac{[}{]}{0pt}{}{\mathbf{x}}{\mathbf{a}_2} \iff \left(\genfrac{[}{]}{0pt}{}{\mathbf{x}}{\mathbf{a}_1} \sim^D_{Q'} \genfrac{[}{]}{0pt}{}{\mathbf{x}}{\mathbf{a}_2} \text{ where } \genfrac{[}{]}{0pt}{}{\mathbf{x}}{\mathbf{a}_1} \wedge F \text{ and } \genfrac{[}{]}{0pt}{}{\mathbf{x}}{\mathbf{a}_2} \wedge F \text{ are satisfiable}\right),$$

$$Q = \exists y.Q'\text{:}\ \text{Let } S_1 = \{b : Q'_1\genfrac{[}{]}{0pt}{}{y\mathbf{x}}{b\mathbf{a}_1} \in \mathsf{PE}_D(Q')\} \text{ and } S_2 = \{b : Q'_2\genfrac{[}{]}{0pt}{}{y\mathbf{x}}{b\mathbf{a}_2} \in \mathsf{PE}_D(Q')\}. \text{ Then}$$
$$\genfrac{[}{]}{0pt}{}{\mathbf{x}}{\mathbf{a}_1} \sim^D_Q \genfrac{[}{]}{0pt}{}{\mathbf{x}}{\mathbf{a}_2} \iff \left((\forall b \in S_1 \exists c \in S_2.\genfrac{[}{]}{0pt}{}{y\mathbf{x}}{b\mathbf{a}_1} \sim^D_{Q'} \genfrac{[}{]}{0pt}{}{y\mathbf{x}}{c\mathbf{a}_2}) \wedge (\forall c \in S_2 \exists b \in S_1.\genfrac{[}{]}{0pt}{}{y\mathbf{x}}{b\mathbf{a}_1} \sim^D_{Q'} \genfrac{[}{]}{0pt}{}{y\mathbf{x}}{c\mathbf{a}_2})\right),$$

$$Q = \exists t.Q'\text{:}\ \text{Let } S_1 = \{s : Q'_1\genfrac{[}{]}{0pt}{}{t\mathbf{x}}{s\mathbf{a}_1} \in \mathsf{PE}_D(Q')\} \text{ and } S_2 = \{s : Q'_2\genfrac{[}{]}{0pt}{}{t\mathbf{x}}{s\mathbf{a}_2} \in \mathsf{PE}_D(Q')\}. \text{ Then}$$
$$\genfrac{[}{]}{0pt}{}{\mathbf{x}}{\mathbf{a}_1} \sim^D_Q \genfrac{[}{]}{0pt}{}{\mathbf{x}}{\mathbf{a}_2} \iff \left((\forall s \in S_1 \exists u \in S_2.\genfrac{[}{]}{0pt}{}{y\mathbf{x}}{s\mathbf{a}_1} \sim^D_{Q'} \genfrac{[}{]}{0pt}{}{y\mathbf{x}}{u\mathbf{a}_2}) \wedge (\forall u \in S_2 \exists s \in S_1.\genfrac{[}{]}{0pt}{}{y\mathbf{x}}{s\mathbf{a}_1} \sim^D_{Q'} \genfrac{[}{]}{0pt}{}{y\mathbf{x}}{u\mathbf{a}_2})\right),$$

$$Q = Q' \wedge Q''\text{:}\ \genfrac{[}{]}{0pt}{}{\mathbf{x}}{\mathbf{a}_1} \sim^D_Q \genfrac{[}{]}{0pt}{}{\mathbf{x}}{\mathbf{a}_2} \iff \left(\genfrac{[}{]}{0pt}{}{\mathbf{x}'}{\mathbf{a}'_1} \sim^D_{Q'} \genfrac{[}{]}{0pt}{}{\mathbf{x}'}{\mathbf{a}'_2} \wedge \genfrac{[}{]}{0pt}{}{\mathbf{x}''}{\mathbf{a}''_1} \sim^D_{Q''} \genfrac{[}{]}{0pt}{}{\mathbf{x}''}{\mathbf{a}''_2},\ \text{for } \genfrac{[}{]}{0pt}{}{\mathbf{x}}{\mathbf{a}_i} = \genfrac{[}{]}{0pt}{}{\mathbf{x}'\mathbf{x}''}{\mathbf{a}'_i\mathbf{a}''_i} \text{ satisfiable}\right),$$

$$Q = Q' \wedge \neg Q'' \text{ or } Q = Q' \vee Q''\text{:}\ \genfrac{[}{]}{0pt}{}{\mathbf{x}}{\mathbf{a}_1} \sim^D_Q \genfrac{[}{]}{0pt}{}{\mathbf{x}}{\mathbf{a}_2} \iff \left(\genfrac{[}{]}{0pt}{}{\mathbf{x}}{\mathbf{a}_1} \sim^D_{Q'} \genfrac{[}{]}{0pt}{}{\mathbf{x}}{\mathbf{a}_2} \wedge \genfrac{[}{]}{0pt}{}{\mathbf{x}}{\mathbf{a}_1} \sim^D_{Q''} \genfrac{[}{]}{0pt}{}{\mathbf{x}}{\mathbf{a}_2}\right).$$

**Figure 2. Equivalence of Abstract Substitutions**

**Definition 4.4** Let $\genfrac{[}{]}{0pt}{}{\mathbf{x}}{\mathbf{a}_1} \sim^D_Q \genfrac{[}{]}{0pt}{}{\mathbf{x}}{\mathbf{a}_2}$ be an equivalence relation on the set of abstract substitutions for free variables $\mathbf{x}$ in the residual queries of $Q$ generated by $\mathsf{PE}_D(Q)$. This relation is defined inductively with respect to the structure of $Q$ using the conditions in Figure 2. Without loss of generality we can assume that abstract substitutions *not* associated with any residual query in the result of $\mathsf{PE}_D(Q)$ are related by $\sim^D_Q$ (the rules for $\vee$ and $\wedge\neg$ take advantage of this fact). □

**Lemma 4.5** *$\sim^D_Q$ is an equivalence relation.* □

This equivalence identifies residual formulas that *behave the same way* in all extensions of $D$ even if they result in *different* tuples (cf. Lemma 4.7). We use the $\sim^D_Q$ equivalence to complete the picture by defining a *time base* for every temporal variable in $Q$—a set of substitutions that are sufficient to produce all answers for a particular subquery:

**Definition 4.6 (Time Base)** Let $Q = \exists t.Q'$ be a query and $\mathbf{a}$ an abstract substitution for $Q$'s free variables. We define a set $\mathsf{TB}_{\mathbf{a}}(t) \subseteq \{s : Q'_1\genfrac{[}{]}{0pt}{}{\mathbf{x}t}{\mathbf{a}s} \in \mathsf{PE}_D(Q')\}$ such that

$$\forall s'.Q'_1\genfrac{[}{]}{0pt}{}{\mathbf{x}t}{\mathbf{a}s'} \in \mathsf{PE}_D(Q') \Rightarrow$$
$$\exists s \in \mathsf{TB}_{\mathbf{a}}(t).Q'_2\genfrac{[}{]}{0pt}{}{\mathbf{x}t}{\mathbf{a}s} \in \mathsf{PE}_D(Q') \wedge \genfrac{[}{]}{0pt}{}{\mathbf{x}t}{\mathbf{a}s'} \sim^D_{Q'} \genfrac{[}{]}{0pt}{}{\mathbf{x}t}{\mathbf{a}s}$$

Note that $\mathsf{TB}_{\mathbf{a}}(t)$ may only contain a single element for every equivalence class of $\sim^D_{Q'}$. □

Note that definitions 4.2, 4.4, and 4.6 mutually depend on each other. However, they are well founded as they can be stratified with respect to the structure of $Q$. The same holds for the following lemmas that establish several properties of the $\mathsf{PE}_D$ operator and the $\sim^D_Q$ relation.

**Lemma 4.7** *Let $D$ be a history, $Q$ a query, $\theta_1, \theta_2$ substitutions for free variables of $Q$, and $\genfrac{[}{]}{0pt}{}{\mathbf{x}}{\mathbf{a}_1} \sim^D_Q \genfrac{[}{]}{0pt}{}{\mathbf{x}}{\mathbf{a}_2}$. Then*

- $(\genfrac{[}{]}{0pt}{}{\mathbf{x}}{\mathbf{a}_1} \supset \genfrac{[}{]}{0pt}{}{x}{\bullet})$ *if and only if* $(\genfrac{[}{]}{0pt}{}{\mathbf{x}}{\mathbf{a}_2} \supset \genfrac{[}{]}{0pt}{}{x}{\bullet})$ *and*
- *if $\theta_1 \models \genfrac{[}{]}{0pt}{}{\mathbf{x}}{\mathbf{a}_1}$ and $\theta_2 \models \genfrac{[}{]}{0pt}{}{\mathbf{x}}{\mathbf{a}_2}$ and whenever $\genfrac{[}{]}{0pt}{}{\mathbf{x}}{\mathbf{a}_1} \supset \genfrac{[}{]}{0pt}{}{x}{\bullet}$ we have $x\theta_1 = x\theta_2$ then*

$$D; D', \theta_1 \models Q \iff D; D', \theta_2 \models Q$$

*for all suffixes $D'$ of $D$.* □

This equivalence identifies residual formulas that behave the same in all extensions of the given history and is used to reduce the number of residual subformulas in the scope of an quantifier and in turn the number of time instants necessary to evaluate the original query:

**Lemma 4.8** *Let $D$ be a history and $Q$ a query. Then*

$$\left(\exists t. \bigvee_{\substack{Q'\genfrac{[}{]}{0pt}{}{\mathbf{x}t}{\mathbf{a}s} \in \mathsf{PE}_D(Q) \\ s \in \mathsf{TB}_{\mathbf{a}}(t)}} Q'\right)\genfrac{[}{]}{0pt}{}{\mathbf{x}}{\mathbf{a}} \equiv \left(\exists t. \bigvee_{Q'\genfrac{[}{]}{0pt}{}{\mathbf{x}t}{\mathbf{a}s} \in \mathsf{PE}_D(Q)} Q'\right)\genfrac{[}{]}{0pt}{}{\mathbf{x}}{\mathbf{a}}.$$

□

With the help of the above Lemmas and Lemma 4.3 the remaining in Figure 1 preserve equivalence between the original query and the set of residual queries. In particular, when handling disjunction and negation, we are guaranteed to have only one residual formula for a given abstract substitution for each of the operands and therefore we do not have to worry about any of the other residual formulas.

**Theorem 4.9 (Equivalence in Extensions)** *Let $D$ be an arbitrary fixed history and $Q$ a query. Then*

$$Q(D; D') = \left(\bigvee_{Q'\genfrac{[}{]}{0pt}{}{\mathbf{x}}{\mathbf{a}} \in \mathsf{PE}_D(Q)} (Q' \wedge \genfrac{[}{]}{0pt}{}{\mathbf{x}}{\mathbf{a}})\right)(D')$$

*for all suffixes $D'$ of $D$.* □

In the rest of the paper we overload the $\mathsf{PE}_D(Q)$ notation to denote the complete residual query in addition to the set of residual queries for particular abstract substitutions.

## 4.2 Residual Database Reconstruction

The result of $\mathsf{PE}_D(Q)$ can be used directly to encode the the *history* of $D$ as follows[4]:

$$\begin{array}{rcl} Q(D) & = & \mathsf{PE}_D(Q)(\emptyset) \\ \mathsf{PE}_{D,D'}(Q) & \equiv & \mathsf{PE}_{D'}(\mathsf{PE}_D(Q)) \end{array}$$

The above approach falls into the category of *query specialization* based techniques for database histories. However, in this paper we look for a *physical deletion* strategy. Therefore we continue to modify the residual query obtained by the $\mathsf{PE}_D$ operator as follows:

**Definition 4.10 (Temporal Support)** Let $t$ be a temporal variable in $Q$ (i.e., $Q$ contains a subformula $\exists t.Q'$). We define a *temporal support of* $t$ to be the set

$$\mathsf{TB}(t) = \bigcup \mathsf{TB}_{\mathbf{a}}(t)$$

where the union ranges over all remaining abstract substitutions generated by $\mathsf{PE}_D(Q)$ for $Q'$. $\Box$

Note that $\mathsf{TB}(t)$ is a valid time base for all abstract substitutions $\mathbf{a}$ (cf. Definition 4.6). This lemma effectively bounds the *useful* valuations of $t$ to the set $\mathsf{TB}(t)$. Therefore every quantifier of the form $\exists t.Q'$ in $\mathsf{PE}_D(Q)$ can be replaced with $\exists t \in \mathsf{TB}(t).Q'$. This formula, however, is equivalent to the original query $Q$ in which the quantifiers are bounded in the same way. Thus we can simply consider these sets, restricted to the current active domain, to be *auxiliary* 0-ary (in the data dimension) relations added to $D$. The remaining step is based on the observation that if all the leaves of $Q$ are of the form $R_i(t_i, \mathbf{x}_i)$ and the valuations of the variables $t_i$ can range only over $\mathsf{TB}(t_i)$ then all the states of $D$ needed to evaluate $Q$ are those in $\bigcup \mathsf{TB}(t_i)$. Since

- the history is unary in the temporal dimension and therefore individual states can be removed independently, and
- the given query is range-restricted and therefore every value for a temporal variable must appear in one of the base relations,

we can remove all other states from the history $D$. Based on the sets $\mathsf{TB}(t_i)$ for every variable $t_i$ in $Q$ (note that due to our restriction to syntactically bounded queries, all these variables are quantified in $Q$) we construct our expiration operator:

**Definition 4.11 (Expiration Operator)** Let $D$ be a history, $Q$ a query and $\mathsf{TB}(t_j)$ time bases for each temporal variable on $Q$. Then we define $\mathcal{E}(D,Q) = Q^{\mathcal{E}}, D^{\mathcal{E}}$ such that:

- $Q^{\mathcal{E}}$ is identical to $Q$ except all subformulas of the form $\exists t_j.Q'$ are replaced by $\exists t_j.\, \mathsf{TB}_{t_j} \wedge Q'$.
- $D^{\mathcal{E}} = \{D_i : D_i \in D, i \in \mathsf{TB}(t_j) \text{ for some } t_j \text{ in } Q\}$; every state $D_i$ is extended with propositional letters $\mathsf{TB}_{t_j}$ defined by $\mathsf{TB}_{t_j} \equiv (i \in \mathsf{TB}(t_j))$. $\Box$

In practice we simply add the relations $\mathsf{TB}_t$ to *every* state of the history and then "run" our expiration process.

**Theorem 4.12 (History Equivalence)** *Let $D$ be a history and $Q$ a range-restricted query. Then (i) $Q(D) = Q^{\mathcal{E}}(D^{\mathcal{E}})$ and (ii) for all suffixes $D'$ of $D$, $(D; D')^{\mathcal{E}}$ can be constructed from $D^{\mathcal{E}}; D'$ using the same expiration operator.* $\Box$

Now we need to consider the size of $D^{\mathcal{E}}$. Since every single state is bounded by a function of the active data domain size, it is sufficient to concentrate on the number of states kept in $D^{\mathcal{E}}$. This number is bounded by the sum of sizes of the $\mathsf{TB}(t_i)$ sets. It is easy to see that the individual sets $\mathsf{TB}_{\mathbf{a}}(t)$ are bounded by the index of the $\sim_Q^D$ relation, which in turn is bounded by a function of the size of the active domain; here we gain up to an exponential factor for every quantifier. Thus $\mathsf{TB}(t_i)$ is bounded by the sum of sizes of $\mathsf{TB}_{\mathbf{a}}(t)$ over all $\mathbf{a}$. As $Q$ is syntactically bounded, the number of different tuples in each subquery can be also bounded by a function of the size of $\mathbf{dom}_D$. All together:

**Theorem 4.13 (Size of the Residual Database)** *Let $D$ be a history and $Q$ a bounded query. Then*

$$|D^{\mathcal{E}}| \leq \min(f(|\mathbf{dom}_D|), |D| + |Q| \cdot |\mathbf{dom}_T|)$$

*where $f$ may contain up to the quantifier depth of $Q$ nested exponentials.* $\Box$

**Example 4.14 (Single Quantifier Alternation)** Consider the query

$$\exists t_1, t_2.t_1 < t_2 \wedge \forall x.P(t_1, x) \iff P(t_2, x).$$

The above query is boolean and can be equivalently expressed as a range-restricted query). It is easy to see that the expired history must keep a copy of every possible subset of the data domain $\mathbf{dom}_D$ that appears in the original history to see if there will be a matching set in the future. On the other hand, it is always sufficient to keep just one such copy and remove all others. $\Box$

The example demonstrates that, in general, we cannot avoid cases where we need to *keep* exponentially many time instants in the size of $\mathbf{dom}_D$.

As a final touch needed to make the expired history $D^{\mathcal{E}}$ truly independent of the length of $D$, we need to consider the binary representation of the *indices* $i$ of $D_i \in D^{\mathcal{E}}$. It is

[4] To prove the second equivalence we extend the $\mathsf{PE}$ operator to handle constants in a natural way; omitted in this paper for sake of simplicity.

again easy to see that for range-restricted queries the actual values of the indices $i$ do not matter as long as the relative order is preserved. Therefore we can renumber the states in $D^{\mathcal{E}}$ using indices with representation bounded by the size of $\mathbf{dom}_D$.

## 5 Implementation Issues

Two simple issues need to be addressed in any implementation of the proposed technique:

1. The *redundancy removal* algorithm in Figure 1 can be implemented using standard first-order updates: for a fixed query $Q$ both the partial evaluation and in turn the sets $\mathsf{TB}(t)$ can be defined using first order-queries: for every $\sim_Q^D$ class we simply pick the minimal time instant with respect to the ordering of time.
2. Temporal databases commonly use an *interval*-based encoding of sets of consecutive time instants, e.g., based on TSQL2 [11]. We have aqlready shown that all well defined queries over such databases can be equivalently expressed over the corresponding point-based databases [3, 12]. Thus our expiration technique can be applied to TSQL-like databases as well, as long as the *append-only* update policy is enforced.

There are several other simple improvements to the basic approach developed in Section 4. In particular:

- the construction of the $\mathsf{TB}(t)$ sets for the individual quantifiers in $Q$ is not coordinated: it is possible to coordinate the choices in a way that maximizes the overlap of the sets and that increases the chance of expiring additional time instants *for all leaves* of $Q$ since this is the only time we may physically remove a state of the warehouse associated with a particular time instant.
- we can use the $\sim_Q^D$ relation to eliminate superfluous data values, similarly to the case of quantification over temporal values (not necessary to obtain the bound on the size of the encoding and therefore omitted in Section 4).
- in the current approach we construct the $\mathsf{TB}(t)$ sets "point-wise": we require every valuation for a variable to be covered by exactly one value in the cover. This condition can be relaxed to having a set of values covered by another set of values.

All the above improvements must be taken with a grain of salt: they are directed towards detecting more time instants that can be removed from the history. However, while the approach outlined in Section 4 can be implemented essentially using first-order queries and therefore fairly efficiently, most of the above improvements lead directly to hard optimization problems and therefore may not be feasible, especially for very large histories.

## 6 Conclusion

We have presented a technique that automatically removes states in a historical data warehouse based on knowing all queries that are to be asked over such a data warehouse in advance. The encoding for histories used by our approach guarantees correctness and, for queries with syntactically bounded answers, is bounded in size by a function of the active *data* domain $\mathbf{dom}_D$ and therefore independent of the length of the history.

While the encoded history is always at most the size of the original history (modulo $|Q||\mathbf{dom}_T|$ bits), the bound derived from the size of the active domain is hyper-exponential and cannot be improved in an expiration-based approach. However, the hyper-exponential bound is only reached if the history itself contains all possible sequences of subsets of the data domain $\mathbf{dom}_D$.

### 6.1 Future Directions

The proposed approach can only handle first-order queries. However, in many warehousing applications, handling aggregation is often necessary.

**Example 6.1** Consider a closed query over a single unary relational scheme that asks "*have there been more $a$'s than $b$'s in the history?*", for $a$ and $b$ two distinct constants. This query needs to maintain the difference of the number of $a$'s and $b$'s, which in turn requires a logarithmic space in the size of the history; this is a lower bound for any technique and follows from the pigeon-hole principle. □

Therefore there is no hope that a technique can maintain queries with aggregation (counting), even when restricted to closed (yes/no) queries, and guarantee that the size of the expired history does not depend on the length of the original history. Similarly to aggregation, allowing even simple retroactive updates makes bounded histories impossible. Consider the following example:

**Example 6.2** Let $D$ be a history containing a single unary relation symbol $R$. Then we execute the following "transaction":

> **while** $\exists t.R(t,a) \land \exists t.R(t,b)$ **do**
>   **delete** $R(t,a)$ **such that** $\forall t'.R(t',a) \supset t' > t$;
>   **delete** $R(t,b)$ **such that** $\forall t'.R(t',b) \supset t' > t$;
> **return** $\exists t.R(t,a)$

It is easy to see that this transaction returns *true* if and only if there have been more $a$'s than $b$'s in the original history $D$. Therefore, using the same argument as in Example 6.1, there cannot be an equivalent history bounded by a function of the size of the active domain. Similar transactions can be exhibited for **insert**s and/or **update**s. □

The proposed technique works for arbitrary first order queries. The space bound, however, can be derived only for *syntactically bounded* queries. The bound cannot apply to *semantically* bounded queries, that can output values of time instants:

**Example 6.3** Consider the query asking for the last time instant in the history. Then, with the progression of time, the single resulting value is logarithmic in the length of the history ($\log(|\mathbf{dom}_T|)$) and therefore not independent of the length of the history. □

We conjecture, however, that in these cases the number of items, disregarding the size of their binary encoding, can still be bounded by a function of the size of the active data domain $\mathbf{dom}_D$. Several other questions are connected with this approach:

- The present paper uses only very simple signatures: equality for the data domain and linear order for the time instants. This alone is sufficient to capture all of first-order temporal logic, or fixed-distance queries. In a practical setting, however, we would probably need on richer interpreted signatures. How does enriching the signatures affect the size and the existence of a bounded encoding?
- So far we have only considered expiration of *complete states* of the given history, although in principle we can expire base relations individually. However, it is not clear if our technique can be applied to expiring individual tuples *within* a particular state. We conjecture that our technique is essentially equivalent to tuple-based expiration for a reified database schema.
- The bound derived for the size of the encoding relies on queries with syntactically-bounded answers. Is there a more general syntactic criterion that captures the class of queries with semantically-bounded answers?
- The approach in this paper can also be extended to maintain aggregation queries by remembering counts of removed tuples in space logarithmic in the length of the history.

In addition, the complexities of an optimal selection of the instants in the $\mathsf{TB}(t_i)$ sets and the other optimizations in Section 5 remain open; to obtain the space bound we used sufficient but possibly sub-optimal approximations.

**Acknowledgments.** The author gratefully acknowledges support by the Natural Sciences and Engineering Research Council of Canada. Part of this work was done while visiting BRICS[5] at the University of Aarhus.

[5]Basic Research in Computer Science (`www.brics.dk`) funded by the Danish National Research Foundation.

## References

[1] S. Abiteboul, L. Herr, and J. Van den Bussche. Temporal Versus First-Order Logic to Query Temporal Databases. In *ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 49–57, 1996.

[2] J. Chomicki. Efficient Checking of Temporal Integrity Constraints Using Bounded History Encoding. *TODS*, 20(2):149–186, 1995.

[3] J. Chomicki and D. Toman. Temporal Logic in Information Systems. In J. Chomicki and G. Saake, editors, *Logics for Databases and Information Systems*, pages 31–70. Kluwer, 1998.

[4] L. S. Colby, T. Griffin, L. Libkin, I. S. Mumick, and H. Trickey. Algorithms for Deferred View Maintenance. In *ACM SIGMOD International Conference on Management of Data*, pages 469–480, 1996.

[5] H. Garcia-Molina, W. Labio, and J. Yang. Expiring Data in a Warehouse. In *International Conference on Very Large Data Bases, VLDB'98*, pages 500–511, 1998.

[6] A. Gupta and I. S. Mumick. Maintenance of Materialized Views: Problems, Techniques, and Applications. *Data Engineering Bulletin*, 18(2):3–18, 1995.

[7] N. Huyn. Multiple-View Self-Maintenance in Data Warehousing Environments. In *International Conference on Very Large Data Bases, VLDB'97*, pages 26–35, 1997.

[8] N. D. Jones, C. K. Gomard, and P. Sestoft. *Partial Evaluation and Automatic Program Generation*. Prentice-Hall International, 1993. Also available online at `http://www.dina.kvl.dk/~sestoft/pebook/pebook.html`.

[9] L. Libkin, G. Kuper, and J. Paredaens, editors. *Constraint Databases*. Springer, 2000.

[10] D. Quass, A. Gupta, I. S. Mumick, and J. Widom. Making Views Self-Maintainable for Data Warehousing. In *Fourth International Conference on Parallel and Distributed Information Systems*, pages 158–169, 1996.

[11] R. T. Snodgrass, editor. *The TSQL2 Temporal Query Language*. Kluwer, 1995.

[12] D. Toman. Point vs. Interval-based Query Languages for Temporal Databases. In *ACM Symposium on Principles of Database Systems*, pages 58–67, 1996.

[13] D. Toman and D. Niwinski. First-Order Queries over Temporal Databases Inexpressible in Temporal Logic. In *Advances in Database Technology, EDBT'96*, volume 1057, pages 307–324. Springer, 1996.

[14] F. W. Tompa and J. A. Blakeley. Maintaining materialized views without accessing base data. *Information Systems*, 13(4):393–406, 1988.

[15] J. Yang and J. Widom. Maintaining Temporal Views over Non-Temporal Information Sources for Data Warehousing. In *Advances in Database Technology, EDBT'98*, pages 389–403, 1998.

[16] J. Yang and J. Widom. Temporal View Self-Maintenance. In *Advances in Database Technology, EDBT'00*, pages 395–412, 2000.