LNCS 1901, pp132-137, September 2000.
(URL: http://www.springer.de/comp/lncs/index.html)

Managing Aging Data Using Persistent Views
(extended abstract)

Janne Skyt   Christian S. Jensen

Department of Computer Science, Aalborg University
`www.cs.auc.dk/~{skyt|csj}`

Abstract. Enabled by the continued advances in storage technologies, the amounts of on-line data grow at a rapidly increasing pace. For example, this development is witnessed in the so-called data webhouses that accumulate data derived from clickstreams. The presence of very large and continuously growing amounts of data introduces new challenges, one of them being the need for effectively managing aging data that is perhaps inaccurate, partly outdated, and of reduced interest. This paper describes a new mechanism, persistent views, that aids in flexibly reducing the volume of data, e.g., by enabling the replacement of such "low-interest," detailed data with aggregated data; and it outlines a strategy for implementing persistent views.

1 Introduction

The developments in data storage technologies continue to obey Moore's Law and thus advance at a rapidly increasing pace. As a consequence, in his recent Turing Award lecture, Jim Gray predicted that there will be sold more disk storage in the 18 months following his lecture than had been sold previously in all of history.

Data storage will be exploited to store increasing amounts of data as soon as it becomes available. The increasing amounts introduce new complexity in data management and offer new challenges to database technology. This paper presents a new mechanism, termed *persistent views* (P-views), that is useful when weeding out data that are no longer desired—e.g., because they are out of date, inaccurate, or are just not needed by any applications—while retaining the data that are still desired.

Consider a sample e-commerce scenario where customer data are accumulated along with web-usage data extracted from clickstreams. The resulting database is effectively append only. The most recent addresses of current customers are used for billing; and geo-coded addresses, recent as well as past, and for current as well as previous customers, are used for data mining. However, only ZIP codes are used in data mining—street names and numbers are of no interest for this task. It is thus desirable to be able to physically delete detailed street data that is not current, or not for current customers. With P-views, one may specify the current, detailed addresses for current customers as a P-view. This ensures that this data is available in the database, even if all address data is deleted from the base relations. As another example, it may be desirable to retain only a high-granularity summary of the web-usage data when this becomes more than one year old. This summary data may be specified as one or several P-views, upon which the detailed access data more than one year old may be physically deleted.

As the context for P-views, the append-only nature of many applications is formalized by introducing relations with transaction-time support [6], for which conventional deletion has only a logical effect. A new mechanism is then needed for physical deletion: we employ vacuuming, which is a particular approach to physical deletion [8]. P-views, views immune to physical deletions on the underlying base relations, are defined in this context. These views enable, in a flexible and user-friendly manner, the retention of, e.g., select, aggregate, or summary data, while also enabling the deletion of detailed data. When data is physically deleted from base relations on which P-views are defined, the base data that is necessary to compute the P-views must be extracted and retained automatically and transparently. A provably correct foundation for accomplishing this extraction has been devised.

The notion of vacuuming was previously presented in [8], which extends and formalizes earlier work by the authors. P-views offer substantial benefits over vacuuming. Next, conventional views, i.e., named and stored query expressions, have been the topic of a multitude of papers. The notion of snapshot, a type of materialized and detached view, was originally advanced by Adiba and Lindsay [1]. In contrast to views, P-views are insensitive to physical deletions, and in contrast to snapshots, P-views are sensitive to insertions and logical deletions. Finally, some work has studied various notions of derived data, often in the context of data warehousing and materialized views [3], [4], [9]. In the most closely related of this work, Garcia-Molina et al. [3] explore how to "expire" (delete) data from materialized views so that a set of predefined, regular views on these materialized views are unaffected and can be maintained consistently with future updates. P-views solve a quite different problem and, e.g., do not involve two levels of views and does not assume a static set of predefined views.

The next section describes the context for this paper. Section 3 defines and characterizes the notion of P-views in general and in relation to vacuuming. Section 4 outlines the implementation of P-views and illustrates their use. Finally, Section 5 summarizes and offers research directions.

2 Context

Many real-world database applications rely on databases that exhibit an append-only behavior. This type of behavior is perhaps best formalized in the notion of a transaction-time database. Consequently, P-views are defined in the context of this type of database, where there are two mechanisms for deleting data, namely logical and physical deletion. Specifically, each tuple has attached two timestamp values, namely a start and an end time. A tuple is part of the current database state at all times in-between its pair of time values.

The interval of a tuple that is currently current extends from some time in the past until the changing current time, which is captured by the variable *now* [2]. A current tuple is logically deleted by replacing its end time of *now* with a fixed value. In contrast, physical deletion of a tuple entails the removal of the tuple from the database.

Consider the following transaction-time relation schema *Person*, recording the states in which people live.

$$Person = \{Name, State\_of\_Residence, Date\_of\_Birth, TT^{\vdash}, TT^{\dashv}\} \tag{1}$$

For simplicity, we use the numbers $1 \ldots 30$ as timestamp values. These may be thought of as days during some specific month. Assuming that three tuples are inserted in the relation at times 2, 5, and 7, we obtain the following relation instance.

$$\{(Jill, DC, 7/6/68, 2, now), (Jim, CA, 1/2/58, 5, now), (Joe, CA, 4/5/63, 7, now)\} \tag{2}$$

Next, at time 12, *Joe* is logically deleted, and *Jill* and *Jim* are physically deleted at times 13 and 16, respectively. Relation $\{(Joe, CA, 4/5/63, 7, 12)\}$ results.

The view below collects the birthday of the youngest person per state, using the aggregation formation operator [5].

$$Young\_pr\_State = \texttt{AGG}_{\{State\_of\_Residence,\ \texttt{MAX}(Date\_of\_Birth)\ \texttt{AS}\ Date\_of\_Birth\}}(Person) \tag{3}$$

This view returns the relation $\{(CA, 4/5/63), (DC, 7/6/68)\}$ from time 7 to 13 and the relation $\{(CA, 4/5/63)\}$ after time 13. The view is affected by the physical deletion; as we shall see next, P-views are not.

3 Persistent Views

The P-view mechanism aims to ease the preservation of summaries of data from a transaction-time database, including data that has been physically deleted, logically deleted, or is current. We define P-views as follows (see reference [7] for a formal definition.)

Definition A P-view, defined at time t on a transaction-time database with physical and logical deletion, evaluates as if no physical deletion had happened at or after time t. □

As a result, even though physical deletion is applied to the base relations, a P-view will always reflect all the data present in the database at any time at or after its definition.

To make the notion of P-views somewhat more specific, we relate it to a particular type of physical deletion, termed vacuuming [8]. Specifically, consider the sample vacuuming specification $\sigma_{TT^{\vdash} < now - 10}(Person)$, entered at time 0, which removes tuples inserted into *Person* more than 10 time units ago. At times 13, 16, and 18, it causes the removal of *Jill*, *Jim*, and *Joe*, respectively.

The view *Young_pr_State* from Section 2, if defined as a P-view at time 12, will evaluate as if all vacuuming on the relation is stopped at this time, i.e., as if the specification is defined as "remove data inserted before time 2."

Therefore the P-view will evaluate to $\{(CA, 1/2/58), (DC, 7/6/68)\}$ even after *Jill* and *Jim* are physically deleted from *Person* at times 13 and 16.

4 Implementation Strategy

The implementation strategy for P-views must incorporate two aspects. First, it must ensure data reduction, meaning that the presence of P-views should not unnecessarily hinder the physical removal of data. Second, the strategy must ensure that P-views remain unaffected by physical removals.

This leads to the question of what data must be preserved and what data must be retained in the system even if it is physically removed from the database. Figure 1 illustrates the implementation strategy we propose. With this strategy,

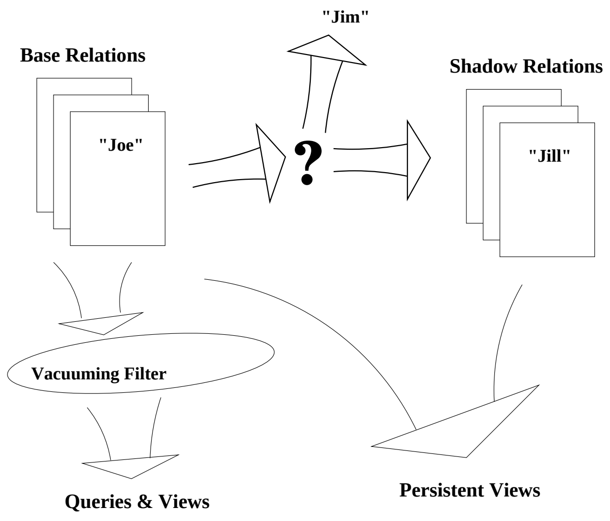

Fig. 1. Implementation Strategy

each base relation is equipped with a shadow relation that has the same schema. The shadow relation retains the tuples that are physically removed from its base relation, but necessary for evaluating a P-view. Further, P-views, unlike regular views and queries, are evaluated on both base and shadow relations.

More specifically, vacuuming separates the logical physical removal of tuples from the actual physical removal by reflecting the vacuuming specifications in a filter. This filter ensures that queries and views do not see tuples that are not yet physically removed, although they should have been so according to the vacuuming specifications. This approach makes it possible to perform the actual physical removal periodically without affecting correctness.

Continuing the *Person* example, the tuples for *Jim* and *Jill* will both qualify for physical removal at time 16. If they have not actually been removed from *Person*, the vacuuming filter will "catch" them so that they appear as physically removed. So with *Young_pr_State* defined as a regular view, the result will correctly be $\{(CA, 4/5/63)\}$.

Next, assume instead that *Young_pr_State* is defined as a P-view. When the tuples for *Jim* and *Jill* are being removed from *Person*, we must decide if the tuples must be retained in order to be able to correctly evaluate the P-view (see the "?" in Figure 1). We determine that the tuple for *Jill* is to be retained in the shadow relation because its absence will affect the P-view result, and that removing the tuple for *Jim* does not affect the result. The order in which the tuples are examined is not significant.

The algorithm given below performs the physical removal and migration to the shadow relation of a set r of tuples.

```
$NewDelete(r, R)$
  [ if $r \neq \emptyset$
    then [select some $u \in r$
          if     $\sigma_{P^R}(R \cup R^S) = \sigma_{P^R}((R \cup R^S) - \{u\})$
          then $Delete(u, R)$;
          else  [$Insert(u, R^S)$; $Delete(u, R)$;]
          $NewDelete(r - \{u\}, R)$;]]
```

Each tuple in r is moved to the shadow relation R^S if its removal affects any P-view; otherwise, the tuple is removed. The second **if**-statement's condition includes a predicate P^R that reflects all existing P-views (reference [7] gives the details).

The implementation strategy outlined above ensures that all views and P-views are always evaluated correctly. Other strategies, e.g., based on materialized views [4], [9], may also be used. The different strategies each have their strengths in different situations.

5 Summary and Research Directions

Motivated by the need for flexible mechanisms to manage the growing amounts of aging data from databases that effectively are append-only, the paper describes a new kind of view, termed *persistent views*, or P-views for short. P-views are similar to conventional views, with the exception that physical deletions on the underlying base relations have no effect on P-views. Although the difference between regular views and P-views is small definition-wise, the implications of this difference are profound. P-views turn out to be quite useful when it is desirable to eliminate bulks of detailed, old, and inaccurate data from the base relations, while preserving only select or aggregate data. In addition, P-views is a general mechanism with applications beyond this paper's focus.

When physically deleting base data, it is generally necessary to retain some of this data transparently to the user in order to be able to compute the P-views. The paper outlines a mechanism for accomplishing this retention using so-called shadow relations.

An extended version of this paper exists [7]. It offers significantly more in-depth coverage of the issues touched upon here.

In future research, it would be of interest to refine the strategy for implementing P-views, so that it retains less data in its shadow relations. This may most prominently be achieved by exploiting projections and, possibly, by introducing multiple shadow relations per base relation. A more radical change would be to abolish the shadow relations altogether and instead use relations that are tied to the individual P-views or subexpressions in P-views. In addition, it would be of interest to prototype the strategy.

Acknowledgments

This research was supported in part by the Danish Technical Research Council through grant 9700780 and by a grant from the Nykredit Corporation.

References

1. M. E. Adiba and B. G. Lindsay. Database Snapshots. In *Proc. VLDB*, pp. 86–91, 1980.
2. J. Clifford, C. Dyreson, T. Isakowitz, C. S. Jensen, and R. T. Snodgrass. On the Semantics of "Now" in Databases. *ACM TODS*, 22(2):171–214, June 1997.
3. H. Garcia-Molina, W. Labio, and J. Yang. Expiring Data in a Warehouse. In *Proc. VLDB*, pp. 500–511, 1998.
4. A. Gupta and I. S. Mumick (editors). *Materialized Views—Techniques, Implementations, and Applications.* The MIT Press, 1999.
5. A. Klug. Equivalence of Relational Algebra And Relational Calculus Query Languages Having Aggregate Functions. *JACM*, 29(3):699–717, July 1982.
6. R. T. Snodgrass and I. Ahn. Temporal databases. *IEEE Computer*, 19(9):35–42, September 1986.
7. J. Skyt and C. S. Jensen. Persistent Views—A Mechanism for Managing Aging Data. TR, Department of Computer Science, Aalborg University, March 2000.
8. J. Skyt, C. S. Jensen, and L. Mark. A Foundation for Vacuuming Temporal Databases. Manuscript under submission.
9. J. Widom (editor). Special Issue on Materialized Views and Data Warehousing. *IEEE Data Engineering Bulletin*, 18(2), June 1995.