Persistent Views—A Mechanism for Managing Ageing Data

JANNE SKYT AND CHRISTIAN S. JENSEN

Aalborg University, Department of Computer Science, Fredrik Bajers Vej 7E, DK-9220 Aalborg Øst, Denmark
Email: skyt@cs.auc.dk

Enabled by the continued advances in storage technologies, the amounts of on-line data grow at a rapidly increasing pace. This development is witnessed in many data warehouse-type applications, including so-called data webhouses that accumulate click streams from portals. The presence of very large and continuously growing amounts of data introduces new challenges, one of them being the need for effective management of aged data. In very large and growing databases, some data eventually becomes inaccurate or outdated and may be of reduced interest to the database applications. This paper offers a mechanism, termed persistent views, that aids in flexibly reducing the volume of data, for example, by enabling the replacement of such 'low-interest', detailed data with aggregated data. The paper motivates persistent views and precisely defines and contrasts these with the related mechanisms of views, snapshots and physical deletion. The paper also offers a provably correct foundation for implementing persistent views.

Received 30 August 2001; revised 6 March 2002

1. INTRODUCTION

Data storage technologies continue to outperform Moore's Law and thus advance at a rapidly increasing pace. As a consequence, Jim Gray, in his Turing Award lecture in June 1999, predicted that there would be more data storage sold in the 18 months following his lecture than had been previously sold in all of history. Also, experience shows that data storage will be exploited to store increasing amounts of data as soon as it becomes available.

While more and more data is stored and exploited over time, some data will eventually become invalid or of only low interest due to changes outside or inside the applications; maybe the reality represented by the data changes or maybe the applications' interests in the data change. Also, laws require the deletion of certain kinds of data after some time, while business policies require the retention of general and aggregated business information. In conclusion, the increasing amounts of data introduce new complexities in data management, which offer new challenges.

This paper presents a new and flexible mechanism termed *persistent views*, or P-views for short, that helps in managing such increasing amounts of data in append-only databases. Briefly, P-views are immune to *physical* deletions. This means that physical deletions from the relations underlying a P-view do not affect the P-view. However, P-views still function as regular views in the context of insertions, (logical) deletions and updates to the underlying relations. P-views offer the following benefits.

- They allow physical deletions to weed out data that no longer is desired while retaining continuously important information. For example, reasons for physically deleting data may be that data is outdated, inaccurate, no longer needed by any applications or that data must be deleted because it is traceable to specific individuals.
- They enable access to anonymous aggregate information based on detailed, possibly traceable data that is to be physically deleted.
- They provide access control, eliminating access to base data.

The paper shows how P-views, in a flexible and user-friendly manner, enable the retention of, for example, 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 and thus render them immune to the deletions is automatically and transparently extracted and retained. The paper offers a provably correct foundation for accomplishing this extraction and thus implementing P-views. (It should be noted, as explained in Section 4, that P-views are independent of the specific mechanism chosen for physical deletion.)

For an illustration, consider click-stream data. It may be desirable to retain only a high-granularity summary of Web-usage data when this reaches a certain age. This summary data may be specified as one or several P-views, upon which the old, detailed access data may be physically deleted. The implementation of P-views ensures that detailed data is reflected correctly in the summary data before it is physically deleted. This and another example application of P-views will be discussed more extensively in the next section.

As the context for P-views, this paper formalizes the append-only nature of many applications by introducing relations with transaction-time support [1, 2]. In these applications, conventional deletions have only a logical effect, so a new mechanism is needed for physical deletion. For this, we employ vacuuming, which is a specific approach to physical deletion. The notion of vacuuming has been studied by a few researchers [3, 4, 5], most recently by Skyt *et al.* P-views offer substantial benefits over vacuuming. Related to vacuuming, Robertson [6] has, at a more abstract level, explored a broad range of issues in relation to the ageing of data and has studied what is termed data transformations.

Next, views, i.e. named and stored query expressions, are fundamental in database management and have been the topic of a multitude of papers. Views are *sensitive to any change* in the underlying database. The notion of a snapshot, a type of materialized and *detached* view, was originally advanced by Adiba and Lindsay [7]. In contrast to snapshots, P-views are sensitive to insertions and logical deletions and in contrast to views, P-views are insensitive to physical deletions.

Some studies of various notions of derived data in an algebraic context exist (see, for example, [8, 9]). Perhaps the most closely related, Garcia-Molina *et al.* [10] 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 different problem and, for example, do not involve two levels of views and do not assume a static set of predefined views.

Finally, in the context of dimensional data warehouses [11], Skyt *et al.* [12, 13] argue that gradual and irreversible aggregation is needed. This also leads to the physical deletion of detail data. Specifically, they suggest an approach for retaining aggregate data, while eliminating detailed data that exploits the hierarchies in dimensions.

The paper is structured as follows. Section 2 presents two example applications of P-views. As a concrete context for defining P-views, Section 3 presents the necessary parts of a temporal data model, a notion of physical deletion, vacuuming and the formal definition of P-views. Section 4 presents a foundation for implementing P-views using so-called shadow relations. Finally, Section 5 summarizes and offers directions for future research. A list of notation, used frequently throughout the paper, can be found in the appendices along with a selection of temporal conventions and definitions.

2. APPLICATIONS OF P-VIEWS

With the purpose of illustrating the utility of P-views in append-only databases, two possible applications are outlined.

The first application relates to click-stream data analysis [14]. Such analysis is increasingly important to e-businesses that want to comprehend their business, including the behavior of their customers [15] or want to produce adaptive Web sites [16, 17].

EXAMPLE 2.1. In click-stream analysis, bulks of click-stream data are accumulated in order to be analyzed. We describe part of a specific click-stream analysis application and illustrate the need for data reduction and the need to retain aggregate information longer than detailed data. Table 1 shows the example data to be used throughout the paper.

Assume that the *news* and *access* relations in Table 1 are part of an e-business application that provides on-line news: user-access information is extracted and combined with news, author and customer information.

News items are represented, by *type of media* (i.e. whether it is plain text, XML, MPG video, audio, etc.), *title*, *author name* and *author address*. The user's *access* to news items is represented by *NewsId*, *UserId* and *Domain* (i.e. from which domain the access occurred). It is recorded when an access took place (in *AccessTime*). Since the system cannot determine when an access terminates, only one time is recorded. For both relations, attributes $TT^{\vdash}$ and $TT^{\dashv}$ record the transaction times of the items. For simplicity, the time unit is a day.

In such an application area, the data growth creates increasing response times to queries. A more serious problem may be that some of the domain values will change. Examples are changes in Web-site structures and Web-page contents (meaning that a page reference represents different content) and users changing physical location or service-providers (which would give the user a new identification). Such changes may adversely affect the meaningfulness of analysis results and better results might occur if the outdated data were physically deleted.

Despite the possible benefits obtained by removing such detailed data, retaining some general aggregated information on this data is still beneficial because it captures the evolution of news accesses over a long period of time. The following expression is a simplified example of such a general aggregation.

$$\mathrm{Agg}_{(\{NewsId,AccessTime\},NoOfAccess,\mathrm{count}(*))}(news \bowtie_{cond} access) \quad (1)$$

where *cond* denotes

$$news.NewsId = access.NewsId \wedge$$
$$news.TT^{\vdash} \leq access.AccessTime \leq news.TT^{\dashv}.$$

The expression makes use of the aggregate formation operator Agg [18]. It partitions the join result on attributes {*NewsId*, *AccessTime*}, introduces a new attribute, *NoOfAccess* and stores in this attribute in each tuple the result of computing count(∗) on the group that the tuple participates in. Also, the aggregate formation operator projects on the grouping attributes and the new aggregate attribute and eliminates duplicates. The query thus computes for each news item and each day the number of accesses to the news item that day.

TABLE 1. Relations *news* and *access* at Time 669.

news

NewsId	*Media*	*Title*	*AuthorName*	*AuthorAddress*	$TT^{\vdash}$	$TT^{\dashv}$
1	Video	Bosnia Today	M. Stone	California	200	*UC*
2	XML	Bill Clinton	I. Cash	New York City	300	*UC*
3	Text	Fall of EU	M. Stone	Washington DC	305	349
4	XML	Fall of EU	M. Stone	Washington DC	350	*UC*

access

AccessId	*NewsId*	*Domain*	*UserId*	*AccessTime*	$TT^{\vdash}$	$TT^{\dashv}$
1	2	aol.com	16	303	303	*UC*
2	2	whitehouse.gov	1201	303	303	*UC*
3	1	aol.com	214	304	304	*UC*
4	1	get2net.dk	512	304	304	*UC*
5	1	eecs.cwru.edu	48	304	304	*UC*
6	1	cs.auc.dk	198	305	305	*UC*
7	3	dr-online.dk	3067	305	305	*UC*
8	3	aol.com	347	305	305	*UC*
9	1	eecs.cwru.edu	12	305	305	*UC*
10	3	aol.com	201	305	305	*UC*

As long as no physical deletion is performed this expression will naturally return the complete result that we are interested in. We now assume that *access* tuples are deleted when their *AccessTime* value is a time point more than 1 year old, e.g. at time 669 the tuples with *AccessId* 1, 2, 3, 4 and 5 will be deleted since their *AccessTime* value is 304 (669 − 365) or less. Then we can no longer be guaranteed to maintain the data needed to compute a complete result to the query.

Therefore, our suggestion is to define the query (1) as a P-view, which would make sure the data necessary to correctly compute the query is retained in the database, while making it possible to delete all the detailed data. Furthermore, the data retained will only be accessible through the P-view. (P-views are formally defined in Section 3.3.)

The second application for P-views is criminal records, i.e. databases recording information about crimes committed, including information about the offenders and their address, social relations, etc.

EXAMPLE 2.2. The availability of criminal records is helpful in the investigation of unsolved crimes and may also help in composing effective crime prevention campaigns. For both purposes, all records are helpful independently of the specifics of the offense and the offender.

Many rules, however, influence what may be kept on record and how the information may be used. Specifically, records of certain minor offenses by juveniles must be removed when the offender reaches legal age. We assume the following schema for relations *Offense*, *Offender* and *Person*:

$$\begin{aligned} \mathit{Offense} &= \{\mathit{OffenseId}, \mathit{Type}, \mathit{OffenseTime}\} \\ \mathit{Offender} &= \{\mathit{OffenseId}, \mathit{SSN}, \mathit{Ruling}\} \\ \mathit{Person} &= \{\mathit{SSN}, \mathit{Name}, \mathit{DateOfBirth}, \mathit{District}\}. \end{aligned}$$

Then all tuples from *Offender* must be deleted whenever the offender is a person with a *DateOfBirth* more than 21 years ago and if all offenses for the person have *Type* 'minor'.

This conflicts with the general interest in crime prevention. For example, access to the records of minor offenses amongst juveniles will reveal which offenses are 'popular', the areas with high concentrations of offenses and the backgrounds of the offenders. The conflicting interests may be managed by applying P-views that offer a variety of non-traceable statistics on minor offenses by year, age, background, location or type of offense. An example is:

$$\mathrm{Agg}_{(\{\mathit{Type},\mathit{District},\mathit{OffenseTime}\},\mathit{NoOfOffenses},\mathrm{count}(*))}(\mathit{Offense} \bowtie \mathit{Offender} \bowtie \mathit{Person}).$$

The result is that non-traceable statistics are available even while the traceable *Person* data used for the aggregations is no longer accessible.

3. DEFINITION OF PERSISTENT VIEWS

We proceed to precisely define P-views. First, we introduce in turn the underlying relation structures and physical deletion.

3.1. Time and temporal relation structures

Let T be a finite, non-empty set of times t with total order $<$. Also let UC (short for 'until changed' [19]) be a variable evaluating to the current time. We use t_{current} for the time in T that corresponds to the current value of UC, and we use T_{UC} to denote the set T including the variable UC. Next, we define the meaning of a time value in T_{UC}.

DEFINITION 3.1. *For times t and t', satisfying $t \in T$ and $t' \in T_{UC}$, the meaning, or value, of t' at time t, $[\![t']\!]_t$ is*

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

Next, assume a set of non-empty domains and a set of attributes. Also, let $TT^{\vdash}$ and $TT^{\dashv}$ be distinguished attributes (capturing transaction time).

DEFINITION 3.2. *A temporal database schema is defined as a finite set of temporal relation schemas. A temporal relation schema is defined as a pair of: (i) a set of user-defined attributes that together with the transaction-time attributes $TT^{\vdash}$ and $TT^{\dashv}$ are the attributes of the schema; and (ii) a function that assigns a domain to each attribute. Specifically it assigns domain T to attribute $TT^{\vdash}$ and domain T_{UC} to attribute $TT^{\dashv}$.*

This definition follows those given in textbooks [20], with the exception that all relation schemas are temporal. The transaction-time attributes $TT^{\vdash}$ and $TT^{\dashv}$ are omni-present. We proceed to define database and relation instances.

DEFINITION 3.3. *A temporal database is a set of temporal relations and a temporal relation is a finite set of tuples. A tuple u in a relation R is a function that, for each attribute A in the schema for R, assigns an element from the domain of A to attribute A. Specifically, it assigns an element in T to $TT^{\vdash}$ and an element in T_{UC} to $TT^{\dashv}$, so that the following constraint is satisfied:*

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

A temporal relation is then a set of tuples, with each tuple being a function assigning values to the attributes such that the interval represented by $TT^{\vdash}$ and $TT^{\dashv}$ starts no later than it ends and such that $TT^{\dashv}$ is UC if $TT^{\vdash}$ is the current time. A tuple u is *current at time t* if and only if $u.TT^{\vdash} \leq t \leq [\![u.TT^{\dashv}]\!]_t$ and simply *current* if it is current at t_{current}.

3.2. Physical deletion

Transaction-time relations are effectively append only, with conventional deletions assuming a purely logical effect. We thus introduce a new facility for physical deletion, termed vacuuming. (Note that P-views are not dependent on the specific choice of physical deletion facility.)

Using vacuuming, specifications are stored in a special temporal relation that expresses what is to be physically deleted. Let *Vspec* be the attribute of this relation that ranges over a domain of vacuuming specification parts, v. This domain contains values of the form '$\omega(R)$: $\sigma_F(R)$', where R is a relation, σ is the standard selection operator and F is a Boolean expression that may involve the attributes of R; ω denotes either ρ or κ, specifying either a removal or a keep specification. The full syntax is given in Appendix B.

A temporal vacuuming specification part, v, is a tuple assigning values from the domain described above to the *Vspec* attribute and values from T and T_{UC} to $TT^{\vdash}$ and $TT^{\dashv}$, respectively. A *temporal vacuuming specification V* is a temporal relation consisting of temporal vacuuming specification parts. Additional detail is offered elsewhere [4, 5].

EXAMPLE 3.1. Continuing Example 2.1, we give both a removal and a keep specification part for relation *access*. In the lifetime of the removal specification part, it specifies removal of the tuples that have an access time (*AccessTime*) that is less than or equal to 365 units before the current time, i.e. tuples timestamped more than 365 time units (days) ago. Similarly, the keep specification part specifies that tuples having a transaction-time end ($TT^{\dashv}$) within the past 183 time units (6 months) must be retained in the relation:

$$\rho(access) : \sigma_{AccessTime \leq UC-365}(access)$$
$$\kappa(access) : \sigma_{TT^{\dashv} > UC-183}(access).$$

The effect of a vacuuming specification V on a relation R is expressed in terms of the set of tuples remaining in R. This set is termed the *vacuumed relation*, denoted by (R, V).

DEFINITION 3.4. *Let $\{v_1, \ldots, v_k, v_{k+1}, \ldots, v_s\}$ be all the specification parts that concern R, i.e. parts with a Vspec value of the form '$\omega(R)$: Exp'. Let $v_i \in \{v_1, \ldots, v_k\}$ be* removal *specification parts and $v_j \in \{v_{k+1}, \ldots, v_s\}$ be* keep *specification parts, where F_i and F_j are the time-dependent predicates of v_i and v_j. Then the vacuumed version of relation R at the current time, (R, V), is defined as*

$$(R, V) \stackrel{\text{def}}{=} \sigma_{\neg(\bigvee_{i=1}^{k} F_i) \vee (\bigvee_{j=k+1}^{s} F_j)}(R).$$

This states that a vacuumed relation is the original relation R where all data selected by any removal specification part ($\rho(R)$) is omitted, except from the data selected by a keep specification part ($\kappa(R)$). For a tuple to be selected by a vacuuming specification part, it must satisfy the selection criteria at some time during the lifetime of the specification part (see [4, 5] for further details).

DEFINITION 3.5. *A vacuumed temporal database is defined as a set of temporal relations vacuumed according to the vacuuming specification.*

3.3. P-views

Sections 1 and 2 briefly stated and illustrated the goals of P-views. Consistent with those discussions, P-views do not interfere with physical deletion, but seen through a P-view, it should appear as if physical deletion has stopped at the time of definition of the P-view.

To define this precisely, assume a database $db = \{R_1, R_2, \ldots, R_n\}$. Then this database at a time t is the set of component relations, each in their respective states, at this time, i.e. $[\![db]\!]_t = \{[\![R_1]\!]_t, [\![R_2]\!]_t, \ldots, [\![R_n]\!]_t\}$ (see Appendix A). $[\![R_i]\!]_t$ denotes the relation R_i as it was at time t, so it contains the tuples inserted into R_i before this time and all tuples that were current at time t have the variable UC as a $TT^{\dashv}$-value. Let V denote the relation specifying the physical deletion.

As specified in Appendix B, a P-view *Pexp* is an expression on one or more relations involving the operators Agg, $\cup$, $\times$, $-$, (), σ and π. The P-view (1) (repeated below) is a specific example of a P-view using this syntax:

$$\mathrm{Agg}_{(\{NewsId, AccessTime\}, NoOfAccess, \mathrm{count}(*))}(news \bowtie_{cond} access). \quad (1)$$

Next, we define the semantics of a P-view.

DEFINITION 3.6. *Let a P-view, given by 'Define P-view P as Pexp', be entered into the database at time $Pexp.TT^{\vdash}$. Then the semantics of Pexp, at time $t \geq Pexp.TT^{\vdash}$ is defined as follows.*

$$[\![Pexp]\!]_t(db, V) \stackrel{\text{def}}{=} Pexp((([\![db]\!]_t, [\![V]\!]_{Pexp.TT^{\vdash}-1}[UC \leftarrow Pexp.TT^{\vdash} - 1]) \cup db_{\text{overhead}}) \quad (2)$$

where db_{overhead} is any set of tuples satisfying the property

$$\forall u' \in db_{\text{overhead}}\ (u' \notin ([\![db]\!]_{Pexp.TT^{\vdash}}, [\![V]\!]_{Pexp.TT^{\vdash}}) \wedge \exists t'\ (t' < Pexp.TT^{\vdash} \wedge u' \in ([\![db]\!]_{t'}, [\![V]\!]_{t'}))). \quad (3)$$

This means that the P-view is defined as the expression *Pexp* evaluated on the union of two elements.

The first element is a database to which some vacuuming is applied. The database db at time t is $[\![db]\!]_t$. The vacuuming applied to this database db is given by $[\![V]\!]_{Pexp.TT^{\vdash}-1}[UC \leftarrow Pexp.TT^{\vdash} - 1]$, which denotes V at time $Pexp.TT^{\vdash} - 1$, where all occurrences of UC are replaced by $Pexp.TT^{\vdash} - 1$. This stops all vacuuming as of this time.

Next, imagine that another P-view was defined prior to the definition of the P-view with expression *Pexp*. This earlier P-view will logically have stopped the vacuuming at the time of its definition. Rather than performing complex bookkeeping, a P-view is simply evaluated also on all such extra data. This is the data in db_{overhead}.

So a P-view is logically evaluated on the current database, where the applied vacuuming is terminated at the time the P-view is defined, combined with (using $\cup$) some data that is retained due to the earlier introduction of other P-views.

EXAMPLE 3.2. Example 2.1 and Table 1 may serve to illustrate the effect of defining a P-view. First, we list the vacuuming specified, assuming there is specified vacuuming for the relation *access* only. Let $V = \{v\}$ where v is

$$(\rho(access) : \sigma_{AccessTime \leq UC-365}(access), 400, UC). \quad (4)$$

This means that the removal specification part is inserted at time 400 and remains current. Assuming the current time is 669, the effect of V is to remove all tuples with an *AccessTime* value less than or equal to 304 ($669 - 365 = 304$), leaving the vacuumed relation $(access, V) = \{6, 7, 8, 9, 10\}$ at this time (we identify each tuple by its *AccessId*).

Next, assume the query in Equation (1) is defined as a P-view at time 500

$$\mathrm{Agg}_{(\{NewsId, AccessTime\}, NoOfAccess, \mathrm{count}(*))}(news \bowtie_{cond} access).$$

Being defined at time 500 this P-view is evaluated on the relations $(news, [\![V]\!]_{499}[UC \leftarrow 499])$ and $(access, [\![V]\!]_{499}[UC \leftarrow 499])$, i.e. it is evaluated as if vacuuming stopped at time 499. Rewinding and terminating the vacuuming specification at time 499 gives the relation V containing the tuple

$$(\rho(access) : \sigma_{AccessTime \leq UC-365}(access), 400, 499). \quad (5)$$

Now vacuuming only removes tuples with *AccessTime*-values less than 134 ($499 - 365 = 134$), and $(access, V) = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$. Thus the P-view will evaluate to the relation $\{(1, 304, 3), (1, 305, 2), (2, 303, 2), (3, 305, 3)\}$.

Had the aggregation query been submitted as a regular query instead, the result would have been based on the vacuumed relation $(access, V) = \{6, 7, 8, 9, 10\}$, resulting in the relation $\{(1, 305, 2), (3, 305, 3)\}$. Thus having defined the P-view, we retain aggregate information in the database.

It should be noted that P-views, as defined and discussed in the prior sections, can only stop the physical deletion that would happen after their introduction. It is not possible to reclaim tuples already physically deleted. This is as intended.

4. IMPLEMENTATION FRAMEWORK

Having defined P-views, we present a framework for implementing them. This framework uses so-called shadow relations, which are presented first. Then the predicates that control which tuples are entered into these shadow relations are derived. Finally, a correctness proof for the framework is described.

4.1. Retaining data in shadow relations

The strategy for implementing vacuuming is to create a filter that hides the tuples that qualify for vacuuming and then to actually physically delete these tuples in an asynchronous/lazy manner [4].

When adding support for P-views, instead of simply physically deleting a tuple, it needs to be evaluated if the tuple is necessary for evaluating a P-view; if so, it is necessary to retain the tuple. For this purpose, we equip each relation with a so-called shadow relation that has the same

schema as that relation. Then P-views must be evaluated on the base relations as well as their shadow relations.

The decision procedure for physical deletion, *NewDelete*, is given next.

DEFINITION 4.1. *Let r be a set of tuples that are to be vacuumed from relation R, let R^S be the shadow relation of R, and let P^R be the predicate (to be specified later) that specifies all tuples that may be necessary for evaluating a P-view. The procedure NewDelete is defined as follows:*

```
PROCEDURE NewDelete (r, R) {
 IF (r ≠ ∅)
 THEN { select some u ∈ r
        IF σ_{P^R}(R ∪ R^S) = σ_{P^R}((R ∪ R^S) − {u})
        THEN { Delete (u, R) }
        ELSE { Insert (u, R^S)
               Delete (u, R) }
        NewDelete (r − {u}, R) } }
```

From the definition of this procedure, it can be observed that P-views are independent of the specific mechanism chosen for physical deletion. The only requirement is that the *NewDelete* procedure is able to examine all tuples that would otherwise be physically deleted.

4.2. Specification of shadow relation predicates

The overall framework for the specification is described, followed by base-step specifications and the remaining specifications for selection and aggregate formation.

4.2.1. *Overall framework*

The definition of *NewDelete* uses the shadow relation predicate P^R on relation R, which specifies the tuples that may be necessary for evaluation of any P-view. This predicate is derived structurally from the P-view expressions that have been defined by the user.

DEFINITION 4.2. *We assume a state s containing a predicate P^R for each relation R in the database. Defining a P-view Pexp affects the predicates P^R. We capture this effect in the* meaning *of an expression Pexp as a partial function on states.*

$$S_{ds} : Pexp \rightarrow (\text{State} \hookrightarrow \text{State}).$$

Initially, all P^R are false. *In Equations* (7)–(28)*, we define the semantics of a P-view expression Pexp in terms of its structure (as specified in Appendix B). We let X, Y and Z be P-view expressions and assume that all selection predicates are in Conjunctive Normal Form (CNF).*

The equations are presented in blocks, and some equations are explained through examples. As a meta-syntax, we use $\star$ to denote *combination*. The application of each basic equation, Equations (7)–(9), essentially adds a disjunct to predicate P^R, rendering the order in which predicates are collected unimportant. Thus, *combination* is symmetric:

$$S_{ds}[\![X \star Y]\!]s = \begin{cases} x' & \text{if } \exists s_1, s_2\ ((S_{ds}[\![X]\!]s = s_1 \wedge S_{ds}[\![Y]\!]s_1 = s') \wedge \\ & \qquad (S_{ds}[\![Y]\!]s = s_2 \wedge S_{ds}[\![X]\!]s_2 = s')) \\ \textit{undefined} & \text{otherwise.} \end{cases} \tag{6}$$

We use $\star$ for combining the semantics of several P-view expressions, independently of the order of definition.

4.2.2. *Base steps for selection and aggregate formation*

We first define the base steps, which have a direct effect on the predicates collected in state s. Each base step introduces predicates stating that tuples satisfying this predicate have to be saved in the shadow relation, to enable a correct evaluation of the P-views. Because tuples satisfying the predicates are indispensable, the predicates are combined with existing predicates using disjunction:

$$S_{ds}[\![R]\!]s = s[P^R \mapsto \text{true}] \tag{7}$$

$$S_{ds}[\![\sigma_p(R)]\!]s = s[P^R \mapsto (P^R) \vee (p)] \tag{8}$$

$$S_{ds}[\![\text{Agg}_{(L,C,func(A_i))}(R)]\!]s = \begin{cases} s[P^R \mapsto (P^R) \vee (min(A_i))] \\ \qquad \text{if } L = \emptyset \wedge func = \text{`}min\text{'} \\ s[P^R \mapsto (P^R) \vee (max(A_i))] \\ \qquad \text{if } L = \emptyset \wedge func = \text{`}max\text{'} \\ s[P^R \mapsto (P^R) \vee (min(A_i \text{ by } B_1, \ldots, B_n))] \\ \qquad \text{if } L = \{B_1, \ldots, B_n\} \wedge func = \text{`}min\text{'} \\ s[P^R \mapsto (P^R) \vee (max(A_i \text{ by } B_1, \ldots, B_n))] \\ \qquad \text{if } L = \{B_1, \ldots, B_n\} \wedge func = \text{`}max\text{'} \\ s[P^R \mapsto \text{true}] \quad \text{otherwise.} \end{cases} \tag{9}$$

As usual, $\sigma_{P^R}(R)$ is all tuples in R that satisfy P^R. When we apply P^R to a tuple, we replace all attribute names A_i in P^R with $u.A_i$. To make Equation (9) well defined, we define the values of the following *min* and *max* expressions for a tuple u in relation R.

$$min(u.A_i) = \begin{cases} \text{true} & \text{if } \forall u' \in R\ (u'.A_i \geq u.A_i) \\ \text{false} & \text{otherwise} \end{cases}$$

$$max(u.A_i) = \begin{cases} \text{true} & \text{if } \forall u' \in R\ (u'.A_i \leq u.A_i) \\ \text{false} & \text{otherwise} \end{cases}$$

$$min(u.A_i \text{ by } u.B_1, \ldots, u.B_n) = \begin{cases} \text{true} & \text{if } \forall u' \in R\ (\forall j \in \{1, \ldots, n\}\ (u'.B_j = u.B_j) \\ & \qquad \Rightarrow u'.A_i \geq u.A_i) \\ \text{false} & \text{otherwise} \end{cases}$$

$$max(u.A_i \text{ by } u.B_1, \ldots, u.B_n) = \begin{cases} \text{true} & \text{if } \forall u' \in R\ (\forall j \in \{1, \ldots, n\}\ (u'.B_j = u.B_j) \\ & \qquad \Rightarrow u'.A_i \leq u.A_i) \\ \text{false} & \text{otherwise.} \end{cases}$$

TABLE 2. Shadow relation $access^S$ after invocation of *NewDelete* at time 680.

AccessId	*NewsId*	*Domain*	*UserId*	*AccessTime*	$TT^{\vdash}$	$TT^{\dashv}$
2	2	whitehouse.gov	1201	303	303	*UC*
9	1	eecs.cwru.edu	12	305	305	*UC*
10	3	aol.com	201	305	305	*UC*

Equations (7)–(9) provide the base cases for implementing P-views via shadow relations. We exemplify case (9).

EXAMPLE 4.1. Using Example 2.1 and Table 1, assume the following P-view is the only P-view defined on the relation *access*:

$$\mathrm{Agg}_{(\{NewsId\},LastAccess,max(AccessTime))}(access).$$

Before defining the P-view, we have that $P^{access} = \text{false}$. Using Equation (9), the set of attributes L to group on is the single attribute *NewsId* and is thus non-empty, and the function is '*max*'. Therefore, $s[P^{access} \mapsto \text{false} \vee max(AccessTime \text{ by } NewsId)]$.

Assume the relation *access* in Table 1, an empty shadow relation $access^S$ and the vacuuming specification (4). Then at time 680, all 10 tuples are chosen for vacuuming, and *NewDelete* completely removes the tuples u satisfying the following equality (having omitted 'false$\vee$' from P^{access}) and moves the rest to the shadow relation:

$$\sigma_{max(AccessTime \text{ by } NewsId)}(access \cup access^S)$$
$$= \sigma_{max(AccessTime \text{ by } NewsId)}((access \cup access^S) - \{u\}).$$

Assuming that *NewDelete* examines the tuples u in the order of their *AccessId* values, this results in the tuples $\{2, 9, 10\}$ being inserted into $access^S$. So, while relation *access* is empty, its shadow relation is as given in Table 2. (The particular evaluation order does not give any advantages or disadvantages in comparison to other orders.)

4.2.3. Remaining base steps

The next part of the base semantics of P-views is expressed in Equations (10)–(14), covering the remaining operators $(\pi, (), \cup, \times, -)$. Each equation defines the semantics of one operator independently of the others:

$$S_{ds}[\![\pi_L(X)]\!]s = S_{ds}[\![X]\!]s \tag{10}$$
$$S_{ds}[\![(X)]\!]s = S_{ds}[\![X]\!]s \tag{11}$$
$$S_{ds}[\![Y \cup Z]\!]s = S_{ds}[\![Y \star Z]\!]s \tag{12}$$
$$S_{ds}[\![Y \times Z]\!]s = S_{ds}[\![Y \star Z]\!]s \tag{13}$$
$$S_{ds}[\![Y - Z]\!]s = S_{ds}[\![Y \star Z]\!]s. \tag{14}$$

Equation (10) accounts for projections in a P-view expression. Since a shadow relation has the same schema as the corresponding base relation and since the result of a projection may still include values from all tuples in the argument X, the equation states that the semantics of projection on X is the same as the semantics of X. Thus, projection does not eliminate any tuples.

EXAMPLE 4.2. Consider the P-view $\pi_{\{Domain\}}(access)$. Assuming $P^{access} = \text{false}$, the application of Equations (10) and (7) yields $S_{ds}[\![\pi_{\{Domain\}}(access)]\!]s = S_{ds}[\![access]\!]s = s[P^{access} \mapsto \text{true}]$. Thus, at time 680 where all 10 tuples in *access* qualify for vacuuming, all 10 tuples will be inserted into $access^S$.

Equation (14) covers the use of set difference in P-views. To calculate a set difference, both the left and the right argument are needed. Thus, the semantics of $Y - Z$ is the semantics of Y combined with the semantics of Z.

EXAMPLE 4.3. The following P-view illustrates Equation (14) and makes use of Equation (8) twice:

$$\sigma_{Domain=\texttt{aol.com}}(access) - \sigma_{NewsId=2}(access).$$

Using the equations and assuming no other P-views, we obtain the following derivation.

$$\begin{aligned} &S_{ds}[\![\sigma_{Domain=\texttt{aol.com}}(access) - \sigma_{NewsId=2}(access)]\!]s \\ &\quad = S_{ds}[\![\sigma_{Domain=\texttt{aol.com}}(access) \star \sigma_{NewsId=2}(access)]\!]s \\ &\quad = S_{ds}[\![\sigma_{NewsId=2}(access)]\!](s[P^{access} \mapsto \text{false} \\ &\qquad\qquad \vee Domain = \texttt{aol.com}]) \\ &\quad = s[P^{access} \mapsto \text{false} \\ &\qquad \vee Domain = \texttt{aol.com} \vee NewsId = 2]. \end{aligned}$$

Referring to Table 1, then if *NewDelete* uses this predicate P^{access} at time 680 where all 10 tuples are vacuumed, *NewDelete* will move the tuples $\{1, 2, 3, 8, 10\}$ into the shadow relation $access^S$.

4.2.4. General steps for selection

The basic equations considered so far define the operators σ and Agg based on a base relation R. Next, we define the semantics of selection in relation to a general expression *Pexp*. We cover the operator in relation to each of the seven operators available. Specifically, Equations (15)–(21) define the semantics for $\sigma_p(Pexp)$:

$$S_{ds}[\![\sigma_{p_1}(\sigma_{p_2}(X))]\!]s = S_{ds}[\![\sigma_{p_1 \wedge p_2}(X)]\!]s \tag{15}$$
$$\begin{aligned} &S_{ds}[\![\sigma_p(\mathrm{Agg}_{(L,C,func(A_i))}(X))]\!]s \\ &\qquad = S_{ds}[\![\mathrm{Agg}_{(L,C,func(A_i))}(X)]\!]s \end{aligned} \tag{16}$$
$$S_{ds}[\![\sigma_p(\pi_L(X))]\!]s = S_{ds}[\![\sigma_p(X)]\!]s \tag{17}$$
$$S_{ds}[\![\sigma_p((X))]\!]s = S_{ds}[\![\sigma_p(X)]\!]s \tag{18}$$
$$S_{ds}[\![\sigma_p(Y \cup Z)]\!]s = S_{ds}[\![\sigma_p(Y) \star \sigma_p(Z)]\!]s \tag{19}$$
$$S_{ds}[\![\sigma_{p_Y \times Z}(Y \times Z)]\!]s = S_{ds}[\![\sigma_{p_Y}(Y) \star \sigma_{p_Z}(Z)]\!]s \tag{20}$$
$$S_{ds}[\![\sigma_p(Y - Z)]\!]s = S_{ds}[\![\sigma_p(Y) \star \sigma_p(Z)]\!]s. \tag{21}$$

In Equation (20), $p_{Y\times Z}$ is required to be in CNF. Thus, $p_{Y\times Z} = p_Y \wedge p_Z \wedge p_{\{Y,Z\}}$, where p_Y, p_Z and $p_{\{Y,Z\}}$ are in CNF. The conjuncts in p_Y only refer to attributes from expression Y, the conjuncts in p_Z only involve attributes from expression Z and those in $p_{\{Y,Z\}}$ involve attributes from both Y and Z.

Equation (16) defines the semantics for a selection based on the result from the aggregate formation operator. This is illustrated next.

EXAMPLE 4.4. Consider the P-view:

$$\sigma_{(NewsId\geq 2)\wedge(LastAccess=303)} \mathrm{Agg}_{(\{NewsId\},LastAccess,max(AccessTime))}(access).$$

Its semantics are defined by Equations (16) and (9):

$$\begin{aligned} &S_{ds}[\![\sigma_{(NewsId\geq 2)\wedge(LastAccess=303)} \mathrm{Agg}_{(\{NewsId\},LastAccess,max(AccessTime))}(access)]\!]s \\ &\quad = S_{ds}[\![\mathrm{Agg}_{(\{NewsId\},LastAccess,max(AccessTime))}(access)]\!]s \\ &\quad = s[P^{access} \mapsto \text{false} \vee max(AccessTime \text{ by } NewsId)] \end{aligned}$$

Using this predicate for P_{access}, *NewDelete* retains the tuples {2, 9, 10} in the shadow relation $access^S$. The P-view then correctly evaluates to {(2, 303)}.

Consider an alternative to Equation (16):

$$\begin{aligned} &S_{ds}[\![\sigma_{p_{Agg}}(\mathrm{Agg}_{(L,C,func(A_i))}(X))]\!]s \\ &\quad = S_{ds}[\![\sigma_{p_X}(X) \star \mathrm{Agg}_{(L,C,func(A_i))}(X)]\!]s, \end{aligned}$$

where p_X denotes the conjunct from p that refers only to attributes in X. Then the example evaluates to $s[P^{access} \mapsto \text{false} \vee NewsId \geq 2 \vee max(AccessTime \text{ by } NewsId)]$, which would result in the tuples {1, 2, 7, 8, 9, 10} being moved to the shadow relation. Thus the P-view would still evaluate to the correct result, but more tuples would be saved in the shadow relations. Thus the semantics in Equation (16) are preferable.

Equation (20) defines the semantics for a selection based on a Cartesian product of two P-view expressions Y and Z. The selection predicate may involve expressions on attributes from Y, Z or on a combination of their attributes. An example follows.

EXAMPLE 4.5. The following P-view illustrates Equation (20):

$$\sigma_{(Domain=\texttt{aol.com}) \wedge (Media=\texttt{XML}) \wedge (access.AccessTime\geq news.TT^{\vdash} \vee news.TT^{\dashv}=UC) \wedge (access.NewsId=news.NewsId)}(access \times news).$$

In this P-view, the base relations *access* and *news* take the role of Y and Z, respectively. Also, the selection predicate is in CNF and has the following three parts:

$$\begin{aligned} p_{access} &= (Domain = \texttt{aol.com}) \\ p_{news} &= (Media = \texttt{XML}) \\ p_{\{access,\, news\}} &= (access.AccessTime \geq news.TT^{\vdash} \\ &\quad \vee news.TT^{\dashv} = UC) \\ &\quad \wedge (access.NewsId = news.NewsId). \end{aligned}$$

Each of these predicates must hold for a tuple from the Cartesian product to be indispensable. Therefore, since the predicate on attributes in *access* only, i.e. p_{access}, does not hold for the tuples {2, 4, 5, 6, 7, 9}, it will not hold for the tuples in the Cartesian product that involve these either. Thus, tuples {2, 4, 5, 6, 7, 9} are disposable, while tuples {1, 3, 8, 10} are possibly indispensable and therefore stored in $access^S$.

Similarly, if vacuuming is defined on base relation *news*, tuples {2, 4} from that relation are possibly indispensable and are thus stored in $news^S$.

A predicate such as

$$(access.AccessTime \geq news.TT^{\vdash} \vee news.TT^{\dashv} = UC)$$

cannot, however, be evaluated based on tuples from one relation alone. The first part clearly involves tuples from both relations. Thus, not all predicates can be used directly on Y or Z to separate the disposable tuples from the indispensable ones.

The semantics of the P-view are derived as follows using Equations (20) and (8):

$$\begin{aligned} &S_{ds}[\![\sigma_{(Domain=\texttt{aol.com}) \wedge (Media=\texttt{XML}) \wedge (access.AccessTime\geq news.TT^{\vdash} \vee news.TT^{\dashv}=UC) \wedge (access.NewsId=news.NewsId)}(access \times news)]\!]s \\ &= S_{ds}[\![\sigma_{Domain=\texttt{aol.com}}(access) \star \sigma_{Media=\texttt{XML}}(news)]\!]s \\ &= S_{ds}[\![\sigma_{Domain=\texttt{aol.com}}(access)]\!] \\ &\quad (s[P^{news} \mapsto \text{false} \vee Media = \texttt{XML}]) \\ &= s[P^{news} \mapsto \text{false} \vee Media = \texttt{XML}, \\ &\quad P^{access} \mapsto \text{false} \vee Domain = \texttt{aol.com}]. \end{aligned}$$

4.2.5. *General steps for aggregate formation*

The last group of equations concern the aggregate formation operator. These are defined in Equations (22)–(28):

$$S_{ds}[\![\mathrm{Agg}_{(L,C,func(A_i))}(\sigma_p(X))]\!]s = S_{ds}[\![\sigma_p(X)]\!]s \tag{22}$$

$$\begin{aligned} &S_{ds}[\![\mathrm{Agg}_{(L_1,C_1,func_1(A_i))}(\mathrm{Agg}_{(L_2,C_2,func_2(A_j))}(X))]\!]s \\ &\quad = S_{ds}[\![\mathrm{Agg}_{(L_2,C_2,func_2(A_j))}(X)]\!]s \end{aligned} \tag{23}$$

$$\begin{aligned} &S_{ds}[\![\mathrm{Agg}_{(L_1,C_1,func_1(A_i))}(\pi_{L_2}(X))]\!]s \\ &\quad = S_{ds}[\![\mathrm{Agg}_{(L_1,C_1,func_1(A_i))}(X)]\!]s \end{aligned} \tag{24}$$

$$\begin{aligned} &S_{ds}[\![\mathrm{Agg}_{(L,C,func(A_i))}((X))]\!]s \\ &\quad = S_{ds}[\![\mathrm{Agg}_{(L,C,func(A_i))}(X)]\!]s \end{aligned} \tag{25}$$

$$\begin{aligned} &S_{ds}[\![\mathrm{Agg}_{(L,C,func(A_i))}(Y \cup Z)]\!]s \\ &\quad = S_{ds}[\![\mathrm{Agg}_{(L,C,func(A_i))}(Y) \star \mathrm{Agg}_{(L,C,func(A_i))}(Z)]\!]s \end{aligned} \tag{26}$$

$$S_{ds}[\![\mathrm{Agg}_{(L,C,func(A_i))}(Y - Z)]\!]s = S_{ds}[\![Y - Z]\!]s \tag{27}$$

$$S_{ds}[\![\mathrm{Agg}_{(L,C,func(A_i))}(Y \times Z)]\!]s = \begin{cases} S_{ds}[\![\mathrm{Agg}_{(L,C,func(A_i))}(Y) \star Z]\!]s \\ \quad \text{if } A_i \in A_Y \wedge L \subseteq A_Y \\ S_{ds}[\![Y \star \mathrm{Agg}_{(L,C,func(A_i))}(Z)]\!]s \\ \quad \text{if } A_i \in A_Z \wedge L \subseteq A_Z \\ S_{ds}[\![Y \times Z]\!]s \quad \text{otherwise.} \end{cases} \tag{28}$$

TABLE 3. Relations $access_1$ and $access_2$.

$access_1$

AccessId	*NewsId*	*Domain*	*UserId*	*AccessTime*	$TT^{\vdash}$	$TT^{\dashv}$
3	1	aol.com	214	304	304	*UC*
4	1	get2net.dk	512	304	304	*UC*
5	1	eecs.cwru.edu	48	304	304	*UC*
6	1	cs.auc.dk	198	305	305	*UC*
7	3	dr-online.dk	3067	305	305	*UC*
8	3	aol.com	347	305	305	*UC*
9	1	eecs.cwru.edu	12	305	305	*UC*
10	3	aol.com	201	305	305	*UC*

$access_2$

AccessId	*NewsId*	*Domain*	*UserId*	*AccessTime*	$TT^{\vdash}$	$TT^{\dashv}$
1	2	aol.com	16	303	303	*UC*
2	2	whitehouse.gov	1201	303	303	*UC*
3	1	aol.com	214	304	304	*UC*
4	1	get2net.dk	512	304	304	*UC*
5	1	eecs.cwru.edu	48	304	304	*UC*

In Equation (28), A_Y is the set of attributes in Y and A_Z is the set of attributes in Z. Note that in Equation (22), also using the semantics from the aggregation would, at most, result in maintaining more tuples in the shadow relations than with the current definition; only tuples in the selection result are used in the aggregation. A similar line of reasoning underlies the design of Equations (23) and (27).

Equation (27) defines the semantics for the aggregate formation operator based on the set difference between two expressions Y and Z. The following example explains.

EXAMPLE 4.6. Consider the following P-view and the relations $access_1$ and $access_2$ shown in Table 3.

$$\mathrm{Agg}_{(\{NewsId\},FirstAccess,min(AccessTime))}(access_1 - access_2).$$

Using Equations (27), (7) and (14) we obtain the predicates P^{access_1} and P^{access_2}:

$$\begin{aligned} &S_{ds}[\![\mathrm{Agg}_{(\{NewsId\},FirstAccess,min(AccessTime))} \\ &\qquad (access_1 - access_2)]\!]s \\ &= S_{ds}[\![access_1 - access_2]\!]s \\ &= S_{ds}[\![access_1 \star access_2]\!]s \\ &= s[P^{access_1} \mapsto \text{true}, P^{access_2} \mapsto \text{true}]. \end{aligned}$$

Thus, all tuples in the two relations will be moved to the shadow relations and evaluation of the aggregate formation operator will be based on the correct set of tuples $\{6, 7, 8, 9, 10\}$ and give the result $\{(1, 305), (3, 305)\}$.

Let us consider the effect of (wrongly!) letting the semantics allow us to take the aggregation into account on Y, Z or both of these. For Y, we would obtain the following equation:

$$\begin{aligned} &S_{ds}[\![\mathrm{Agg}_{(L,C,func(A_i))}(Y - Z)]\!]s \\ &\qquad = S_{ds}[\![\mathrm{Agg}_{(L,C,func(A_i))}(Y) \star Z]\!]s. \end{aligned}$$

Y: Allowing the basic semantics on Y, i.e. $access_1$, we would get

$$s[P^{access_1} \mapsto min(AccessTime \text{ by } NewsId), \\ P^{access_2} \mapsto \text{true}].$$

This will result in the shadow relations $access_1^S = \{5, 10\}$ and $access_2^S = \{1, 2, 3, 4, 5\}$. The P-view would then evaluate to $\{(3, 305)\}$, since the set difference will return only tuple 10.

Z: Allowing the basic semantics on Z, i.e. $access_2$, we would get

$$s[P^{access_1} \mapsto \text{true}, \\ P^{access_2} \mapsto min(AccessTime \text{ by } NewsId)].$$

This will result in the shadow relations $access_1^S = \{3, 4, 5, 6, 7, 8, 9, 10\}$ and $access_2^S = \{2, 5\}$. The P-view evaluates to $\{(1, 304), (3, 305)\}$, based on the tuples $\{4, 10\}$.

Y/Z: Allowing the basic semantics on both $access_1$ and $access_2$, we would get

$$s[P^{access_1} \mapsto min(AccessTime \text{ by } NewsId), \\ P^{access_2} \mapsto min(AccessTime \text{ by } NewsId)].$$

This will result in the shadow relations $access_1^S = \{5, 10\}$ and $access_2^S = \{2, 5\}$. The P-view now evaluates to $\{(3, 305)\}$, based on tuple 10.

Thus, the aggregate formation operator gives no useful knowledge regarding whether tuples from either $access_1$ or $access_2$ are disposable or indispensable.

4.3. Correctness

The theorem that follows states that the proposed mechanism for accumulating data in shadow relations enables the system to correctly compute P-views.

THEOREM 4.7. *Assume that the NewDelete-procedure is used for removing vacuumed data, as described throughout Sections 4.1 and 4.2. Then for all vacuuming specifications V, databases db, P-views $Pexp$ and times $t \geq Pexp.TT^{\vdash}$, the following holds:*

$$Pexp([\![db]\!]_t \cup [\![db^S]\!]_t) = [\![Pexp]\!]_t(db, V). \quad (29)$$

Proof. Applying the definition of P-views (Definition 3.6) to the right-hand side of Equation (29), we see that to prove Theorem 4.7 we need to *show the existence of an overhead* db_{overhead} such that Equation (29) is satisfied.

We choose $db_{\text{overhead}} = \{R_1^O, \ldots, R_n^O\}$, where each R_i^O has the same schema as a relation R_i in db and where

$$\begin{aligned} R_i^O = \{u \mid \exists t' < Pexp.TT^{\vdash}\ (u \in ([\![R_i]\!]_{t'-1}, [\![V]\!]_{t'-1}) \\ \wedge\ u \notin ([\![R_i]\!]_{t'}, [\![V]\!]_{t'}) \wedge\ \sigma_{[\![P^{R_i}]\!]_{t'}}(R_i \cup R_i^S) \\ \neq \sigma_{[\![P^{R_i}]\!]_{t'}}((R_i \cup R_i^S) - \{u\}))\}. \end{aligned} \quad (30)$$

Here, $([\![R_i]\!]_{t'}, [\![V]\!]_{t'})$ and $([\![R_i]\!]_{t'-1}, [\![V]\!]_{t'-1})$ denote the relation R_i vacuumed at time t' and $t'-1$, respectively. The predicate $[\![P^{R_i}]\!]_{t'}$ is the shadow relation predicate for relation R_i constructed from the P-views at time t' as specified in Definition 4.2.

To see that this is a valid choice, note that for any tuple $u \in db_{\text{overhead}}$ there exists a corresponding relation R_i from where u was removed at time t', i.e. before the time $Pexp.TT^{\vdash}$. Therefore, u is not there at time $Pexp.TT^{\vdash}$, but was there at an earlier time $t'-1$. Thus, db_{overhead} is a set of tuples satisfying the property stated in Definition 3.6. Furthermore, as chosen, a tuple u in db_{overhead} was vacuumed at time t' and because $\sigma_{[\![P^{R_i}]\!]_{t'}}(R_i \cup R_i^S) \neq \sigma_{[\![P^{R_i}]\!]_{t'}}((R_i \cup R_i^S) - \{u\})$, i.e. the criteria used by *NewDelete*, u was saved in the shadow relation. Therefore, db_{overhead} is the exact set of tuples saved in db^S before time $Pexp.TT^{\vdash}$ due to the existence of P-views.

Having chosen db_{overhead} like this, we show Theorem 4.7 in two steps.

Step 1. For the argument databases on the left- and right-hand sides of Equation (29) (having substituted the right-hand side by the definition of P-views, Definition 3.6) show the inclusion:

$$\begin{aligned} &([\![db]\!]_t \cup [\![db^S]\!]_t) \\ &\quad \subseteq (([\![db]\!]_t, [\![V]\!]_{Pexp.TT^{\vdash}-1}[UC \leftarrow Pexp.TT^{\vdash} - 1]) \\ &\qquad \cup\ db_{\text{overhead}}). \end{aligned} \quad (31)$$

This shows that the physical database achieved by using the shadow relations is contained in the database used in Definition 3.6.

Step 2. If we denote the tuples that appear on the right-hand side but not on the left-hand side of inclusion (31) as *excess tuples*, then show that no excess tuple u will influence the evaluation of the P-view, i.e. that

$$\begin{aligned} &u \in (([\![db]\!]_t, [\![V]\!]_{Pexp.TT^{\vdash}-1}[UC \leftarrow Pexp.TT^{\vdash} - 1]) \\ &\qquad \cup\ db_{\text{overhead}}) \\ &\wedge\ u \notin ([\![db]\!]_t \cup [\![db^S]\!]_t) \\ &\Downarrow \\ &Pexp(([\![db]\!]_t, [\![V]\!]_{Pexp.TT^{\vdash}-1}[UC \leftarrow Pexp.TT^{\vdash} - 1]) \\ &\qquad \cup\ db_{\text{overhead}}) \\ &= Pexp((([\![db]\!]_t, [\![V]\!]_{Pexp.TT^{\vdash}-1}[UC \leftarrow Pexp.TT^{\vdash} - 1]) \\ &\qquad \cup\ db_{\text{overhead}}) - \{u\}). \end{aligned} \quad (32)$$

First, all tuples in db_{overhead} also belong to $[\![db^S]\!]_t$, so no tuple in db_{overhead} is an excess tuple.

Using this, we show that all excess tuples have the property

$$\begin{aligned} &\exists t' \geq Pexp.TT^{\vdash}\ (\neg Pvac_{t'-1}(u) \wedge Pvac_{t'}(u) \\ &\quad \wedge \sigma_{[\![P^R]\!]_{t'}}(R \cup R^S) = \sigma_{[\![P^R]\!]_{t'}}((R \cup R^S) - \{u\})). \end{aligned} \quad (33)$$

This property states that an excess tuple is vacuumed at a time t' after time $Pexp.TT^{\vdash}$ and that the *NewDelete* procedure, based on the shadow relation predicate P^R which was collected for relation R as explained throughout Section 4.2, did not save it in a shadow relation.

Finally, using induction in the structure of the predicate P^R, we show implication (32), i.e., that no excess tuple influences the evaluation of the P-view, *Pexp*.

The two steps prove that choosing db_{overhead} as above in Equation (30), a tuple removed from the database by the *NewDelete* procedure (within the lifetime of a P-view) has no influence on the evaluation of the P-view. This shows that the implementation strategy is correct. □

A detailed proof, containing the proofs of Step 1 and Step 2, is presented in the technical report [21].

4.4. Relationship to views and snapshots

Having defined P-views and an accompanying implementation framework, we illustrate how P-views differ from traditional views and snapshots, and we emphasize the independence of P-views on the specific mechanism chosen for physical deletion. We use Example 2.1 to explore how the need for physical deletion and for summary data may be accommodated simultaneously using vacuuming in conjunction with existing mechanisms such as either traditional views or snapshots [7].

First, assume a traditional view is used in place of the P-view. Then the physical deletion mechanism (i.e.

vacuuming) needs to be adjusted to enable computing the view correctly; otherwise, the view is affected and information is lost when vacuuming occurs (see the example below). To avoid such a loss, new keep specifications may be entered or removal specifications may be changed. However, this is not an attractive solution, since the adjustments tend to be very difficult, involving complicated specifications. Alternatively, the vacuuming will end up very 'imprecise', leading to little actual physical removal. Also, the potential for physical removal is compromised, since a correct computation of a traditional view is based on the base relations. Finally, the legal requirements for removing data are not met—the extra data will be retained in the base relations and will be accessible at least to the database administrator. In conclusion, combining vacuuming and traditional views is inadequate.

EXAMPLE 4.8. Assume that the P-view with schema $\{NewsId, AccessTime, NoOfAccess\}$, specified in Equation (1), is created at time 350 and evaluates to $\{(1, 304, 3), (1, 305, 2), (2, 303, 2), (3, 305, 3)\}$. Assume that the vacuuming specification in Equation (4) takes effect at time 400. At this moment, only tuples having their *AccessTime* value less than or equal to 35 $(400 - 365)$ are removed, so no tuples are removed, and the view is intact. However, at time 669, the tuples with *AccessTime* value less than or equal to 304 are absent. Five tuples are affected (see Table 1) and when the view is recomputed, only the five tuples remaining in *access* will be considered, yielding the resulting relation $\{(1, 305, 2), (3, 305, 3)\}$. We have thus lost the desired access to summary data.

As another alternative, assume that a snapshot [7] is used in place of the P-view. The snapshot comprises a static picture of the answer to the query expression at the time of its creation. Even after introducing vacuuming, the snapshot will remain unchanged as desired. The problem is that when a new tuple is inserted into the relation, this tuple will not be reflected in the result. Also, creating the snapshot again will not produce the correct result since data has been vacuumed.

In conclusion, traditional views and snapshots fall short in meeting the application's needs. The proposed P-views aim to meet these unmet needs.

5. SUMMARY AND RESEARCH DIRECTIONS

Motivated by the need for flexible mechanisms to manage the growing amounts of aged or obsolete data and based on the observation that a wide range of applications—e.g. financial and medical applications and applications in e-business and data warehousing—rely on append-only databases, the paper introduces a new kind of view, termed *persistent views*, or P-views for short.

In append-only databases, deletion has a logical effect only; all past database states are retained, with the result that data volumes grow monotonically. New physical deletion facilities, termed vacuuming, are introduced. P-views are similar to conventional views, with the exception that physical deletions have no effect on P-views. Although definition-wise the difference between regular views and P-views is small, the implications of this difference are profound. We emphasize the following:

- P-views allow one to delete and weed out the detailed data while retaining select and aggregate information.
- P-views enable access to anonymous aggregate informations, when deletion of the detailed data is required.
- P-views offer access control on detailed data.

The paper shows how P-views are quite useful for eliminating bulks of detailed, old and inaccurate data, while preserving only select or aggregate data. Specifically, one may specify P-views that retrieve the desired, e.g. aggregate, data from the base relations and then physically delete all detailed data from the base relations. In addition, P-views is a general mechanism that has applications beyond the focus of this paper.

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 proposes a mechanism for accomplishing this retention using so-called shadow relations, thus offering a systematic approach to implementing P-views.

In future research, it would be of interest to further refine the foundation for implementing P-views, since the current foundation retains more data in its shadow relations than is strictly necessary for computing the P-views. Most prominently, projections in P-views are not exploited to retain less data. This may possibly be achieved 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 of P-views. This may provide better chances of storing less data, while still being able to compute P-views correctly.

In addition, it would be of interest to prototype the foundation and investigate performance issues relevant to the implementation strategy. This would support research on a number of important questions related to system architecture, minimum DBMS requirements and storage overhead.

REFERENCES

[1] Jensen, C. S. and Lomet, D. B. (2001) Transaction timestamping in (temporal) databases. In *Proc. 27th Int. Conf. on Very Large Databases*, Roma, Italy, 11–14 September, pp. 441–450. Morgan Kaufmann Publishers, Orlando, FL.

[2] Snodgrass, R. T. and Ahn, I. (1986) Temporal databases. *IEEE Computer*, **19**, 35–42.

[3] Lomet, D. B. and Salzberg, B. (1993) Transaction-time databases. In A. U. Tansel *et al.* (eds), *Temporal Databases: Theory, Design, and Implementation*, pp. 388–417. Benjamin-Cummings, Redwood City, CA.

[4] Skyt, J. and Jensen, C. S. (1998) *Vacuuming Temporal Databases*. TIMECENTER Technical Report TR-32, Aalborg University, Aalborg, Denmark. www.cs.auc.dk/TimeCenter/pub.htm

[5] Skyt, J., Jensen, C. S. and Mark, L. (2002) A foundation for vacuuming temporal databases. To appear in *Data and Knowledge Engineering*. Elsevier Science, Netherlands.

[6] Robertson, C. J. (1993) *The Temporal Transformational Data Model*. PhD Thesis, University of Otago, Dunedin, New Zealand.

[7] Adiba, M. E. and Lindsay, B. G. (1980) Database snapshots. In *Proc. 6th Int. Conf. on Very Large Databases*, Montreal, Quebec, 1–3 October, pp. 86–91. IEEE Computer Society Press, Los Alamitos, CA.

[8] Gupta, A. and Mumick, I. S. (eds) (1999) *Materialized Views—Techniques, Implementations, and Applications*. MIT Press, Cambridge, MA.

[9] Widom, J. (ed.) (1995) Special issue on materialized views and data warehousing. *IEEE Data Eng. Bull.*, **18**, 3–47.

[10] Garcia-Molina, H., Labio, W. and Yang, J. (1998) Expiring data in a warehouse. In *Proc. 24th Int. Conf. Very Large Databases*, New York City, NY, 24–27 August, pp. 500–511. Morgan Kaufmann Publishers, San Francisco, CA.

[11] Inmon, W. H. (2001) *Corporate Information Factory* (2nd edn). John Wiley & Sons, New York.

[12] Skyt, J. and Jensen, C. S. (2000) Managing aging data using persistent views (extended abstract). In Etzion, O. and Scheuermann, P. (eds), *Proc. 7th Int. Conf. Cooperative Information Systems*, Eilat, Israel, 6–8 September. *Lecture Notes in Computer Science*, **1901**, 132–137. Springer, Berlin.

[13] Skyt, J., Jensen, C. S. and Pedersen, T. B. (2001) *Specification-Based Data Reduction in Dimensional Data Warehouses*. TIMECENTER Technical Report TR-61, Aalborg University, Aalborg, Denmark. www.cs.auc.dk/TimeCenter/pub.htm

[14] Kimball, R. and Merz, R. (2000) *The Data Webhouse Toolkit*. John Wiley & Sons, New York.

[15] Andersen, J., Giversen, A., Jensen, A. H., Larsen, R. S., Pedersen, T. B. and Skyt, J. (2000) Analyzing clickstreams using subsessions. In *Proc. 3rd Int. Workshop on Data Warehousing and OLAP*, McLean, VA, 10 November, pp. 25–32. ACM, New York, NY.

[16] Perkowitz, M. and Etzioni, O. (1997) Adaptive sites: automatically learning from user access patterns. *Computer Networks (and ISDN Systems)*, **29**. *Papers from the 6th Int. World Wide Web Conf.*, Santa Clara, CA, 7–11 April. Elsevier Science, Netherlands. Poster no. 722. www.scope.gmd.de/info/www6/posters/722/index.html.

[17] Perkowitz, M. and Etzioni, O. (1999) Towards adaptive web sites: conceptual framework and case study. *Computer Networks (and ISDN Systems)*, **31**. *Proc. 8th Int. World Wide Web Conf.*, Toronto, Canada, 11–14 May, pp. 1245–1258. Elsevier Science, Netherlands.

[18] Klug, A. (1982) Equivalence of relational algebra and relational calculus query languages having aggregate functions. *J. ACM*, **29**, 699–717.

[19] Clifford, J., Dyreson, C. E., Isakowitz, T., Jensen, C. S. and Snodgrass, R. T. (1997) On the semantics of 'Now' in databases. *ACM Trans. Database Syst.*, **22**, 171–214.

[20] Abiteboul, S., Hull, R. and Vianu, V. (1995) *Foundations of Databases*. Addison-Wesley, Reading, MA.

[21] Skyt, J. and Jensen, C. S. (2001) *Persistent Views—A Mechanism for Managing Aging Data*. TIMECENTER Technical Report TR-65, Aalborg University, Aalborg, Denmark. www.cs.auc.dk/TimeCenter/pub.htm

APPENDIX A. THE STATE OF A TEMPORAL RELATION

The state of relation R_i at time t, $[\![R_i]\!]_t$ is formally defined as follows.

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

Here, $u \stackrel{v.e.}{=} u'$ is true if u and u' are value equivalent, i.e. the attributes of u and u' are mutually identical except for the transaction-time values. Thus, the state of R_i at time t is the set of tuples from R_i inserted before t and having UC as transaction-time end ($TT^{\dashv}$) if not yet deleted at time t.

APPENDIX B. SYNTAX SPECIFICATIONS

The table below presents the syntax of a vacuuming specification part (v) and a P-view (*Pexp*). Specifically for P-view *Pexp*, L is a list of attribute names, C is a name given to the aggregate attribute achieved by using the aggregate function *func* grouped on L, and A_i denotes an attribute name.

First we present a few preliminary conventions and definitions.

Let a time interval be a pair of times (t_1, t_2) where $t_2 > t_1$. We define a time span s, or duration, to be an unanchored time interval, i.e. a 'time interval' with no specific start or end time, but only a length (2 *months* and 5 *units* are examples of time spans).

Also, let S be a finite, non-empty set of time spans, D_{A_i} be the value domain for an attribute A_i and U_A be the set of attributes A_i.

Pexp	::=	$\text{Agg}_{(L,C,func(A_i))}(Pexp) \mid Pexp \cup Pexp \mid Pexp \times Pexp \mid Pexp - Pexp \mid (Pexp) \mid \sigma_F(Pexp) \mid \pi_L(Pexp) \mid R$
L	::=	$\{List\} \mid \emptyset$
List	::=	$A_i, List \mid A_i$
v	::=	$\omega(R) : Exp$
ω	::=	$\rho \mid \kappa$
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	::=	$t \mid s \mid tt - tt \mid tt + tt \mid (tt)$
TT	::=	$TT^{\vdash} \mid TT^{\dashv}$
bop	::=	$\vee \mid \wedge$
op	::=	$< \mid > \mid = \mid \leq \mid \geq \mid \neq$

Some conventional semantic constraints must also be followed in addition to the specified syntax. It is required that $d \in D_{A_i}$, $A_i \in U_A$, $t \in T_{UC}$ and $s \in S$. For expressions such as $TT\ op\ tt$ and $tt\ op\ TT$, op should be defined for the domain T_{UC} of $TT^{\vdash}$ and $TT^{\dashv}$, and tt should evaluate to an element in T_{UC}. 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 $d \in D_{A_i}$. Finally, for $\sigma_F(Exp)$, F should only include attributes A_i in *Exp*.

APPENDIX C. FREQUENTLY USED NOTATION

This appendix collects and explains the notation used frequently throughout the paper.

UC	*Time variable.* A variable evaluating to the current time. UC is the long name.
T, T_{UC}	*Sets of timestamp values.* T is finite and non-empty, and $T_{UC} = T \cup \{UC\}$.
$TT^{\vdash}$, $TT^{\dashv}$, *AccessTime*, A_i, *Vspec*	*Attribute names.* TT denotes transaction time, and *AccessTime* denotes valid time; symbols $\vdash$ and $\dashv$ denote the start and end of an interval, respectively. Symbol A_i is used for non-specific attributes, and *Vspec* is the specific attribute in the vacuuming relation that stores the specification of vacuuming.
t, t_i, t', t_{def} t_{current}	*Timestamps.* t_{current} denotes the current time.
u, u', u_i, v, v', v_i, v_j	*Tuples.* The u's are used for tuples in general, and the v's are used for the tuples that are vacuuming specification parts.
$v.Vspec$, $u.A_i$, $u.TT^{\vdash}$	*Attribute values.* ⟨tuple⟩.⟨attribute⟩ is generally used to express the value of attribute ⟨attribute⟩ for the ⟨tuple⟩.
R, R_i, V	*Relations.* V is the relation storing information on vacuuming specifications.
(R, V)	*Vacuumed relation.* (R, V) denotes the relation R vacuumed by the specification present in V.
db	*Database.*
R^S, db^S	*Shadow structures.* R^S is the shadow relation for relation R, which is used for evaluating P-views on R. db^S denotes the set of shadow relations for the relations in db.
F, F_i	*Selection predicates.* Upper-case letters are used for predicates in vacuuming specifications.
P^R	Shadow relation predicate for relation R. This predicate is used for deciding if a tuple is needed in R^S.
p, p_1, p_R, p_X	Lower-case p's are generally used in the expressions defining P-views.
$[\![u.TT^{\dashv}]\!]_t$, $[\![R]\!]_t$, $[\![db]\!]_t$	*Evolving structures.* The notation $[\![X]\!]_t$ applies to an evolving structure X and it returns X as it was at time t. Specifically, $[\![u.TT^{\dashv}]\!]_t$ denotes the value of the transaction time end attribute for tuple u as it was at time t. Likewise, $[\![R]\!]_t$ and $[\![db]\!]_t$ denote the *state* of R and db as they appeared at time t.
$X[x \leftarrow y]$	*Value replacement.* Denotes X where all occurrences of x are replaced by y.
$s[x \mapsto y]$	*Denotational semantics.* Notation for the state s extended with the assignment of y to x.
$S_{ds}[\![Pexp]\!]s$	Notation for the effect on state s of introducing P-view *Pexp*.