An Incremental Algorithm to Optimally Maintain Aggregate Views

Abhijeet Mohapatra and Michael Genesereth

Stanford University
{abhijeet, genesereth}@stanford.edu

Abstract

We propose an algorithm called *CReaM* to incrementally maintain materialized aggregate views with user-defined aggregates in response to changes to the database tables from which the views are derived. We show that when the physical design of the underlying database is optimized, the time taken by CReaM to update an aggregate view is *optimal*.

1 Introduction

In data management systems, views are derived relations that are computed over database tables (which are also known as extensional database relations or *edbs*). Views are materialized in a database to support efficient querying of the data. A materialized view becomes out-of-date when the underlying edb relations from which the view is derived are changed. In such cases, the materialized view is either recomputed from the edb relations or the changes in the edb relations are *incrementally* propagated to the view to ensure the correctness of the answers to queries against the view. Prior work presented in [2, 13, 18] shows that incrementally maintaining a materialized view can be significantly faster than recomputing the view from the edb relations especially if the size of the view is large compared to the size of the changes.

A survey of the view maintenance techniques is presented in [7]. However, only a small fraction of the prior work on incremental view maintenance [6, 8, 9, 13, 15, 18, 19] addresses the maintenance of views that contain aggregates such as sum and count. The techniques proposed in [6, 8, 13, 15] incrementally maintain views that have only one aggregation operator. Furthermore, the incremental maintenance algorithms presented in [6, 8, 9, 13, 15, 18, 19] support only a fixed set of built-in aggregate operators (min, max, sum, and count). In contrast, we present an algorithm to incrementally maintain views with *multiple* aggregates each of which could be *user-defined*.

The paper is organized as follows. In Section 2, we discuss the specification of aggregates in our language. In Section 3, we present differential rules to correctly characterize the changes in edb relations to aggregate views. In Section 4, we present an incremental algorithm called *CReaM* that leverages the differential rules to *optimally* maintain materialized aggregate views. In Section 5, we compare our work to prior work on incremental maintenance of aggregate views.

Before we discuss our proposed solution, we illustrate the problem of incrementally maintaining aggregate views using a running example.

Running Example: Suppose that there are tournaments in the Star Wars universe on different planetary systems. The tournament results are recorded in an edb relation, say tournament(V, D, L). A tuple $(V, D, L) \in$ tournament *iff* V has defeated D on the planet L. For instance, if Yoda has defeated Emperor Palpatine at Dagobah then the tuple (yoda, palpatine, dagobah) is in the extension of the edb relation tournament. We use the extension of tournament that is presented in Table 1 in examples throughout the paper.

We define and materialize a view, say victories(V, W) to record the number of victories W achieved by a character V on a planet. For instance, Yoda has two victories in Dagobah and one victory in

tournament		
Victor	**Defeated**	**Location**
yoda	vader	dagobah
yoda	palpatine	dagobah
vader	yoda	tatooine
yoda	palpatine	tatooine

victories	
Victor	**Wins**
yoda	2
vader	1
yoda	1

Table 1: Extension of the edb relation *tournament* and the view *victories* in the Star Wars Universe

Tatooine. Therefore the tuples (yoda, 2) and (yoda, 1) $\in$ victories. The extension of victories that corresponds to the extension of the edb relation tournament is presented in Table 1.

Suppose that a new tournament match is played in Tatooine and that Darth Vader defeats Emperor Palpatine in this match. The new tournament match at Tatooine causes an *insert* to the tournament relation. In response to the insert to the tournament relation, the tuple (vader, 1) $\in$ victories must be updated to (vader, 2) to ensure the correctness of the answers to queries against the view. Now, suppose that the previous tournament match between Yoda and Palpatine at Tatooine is invalidated. In this case, the tuple (yoda, palpatine, tatooine) is *deleted* from the tournament relation. In response to this deletion, the tuple (yoda, 1) must be deleted from the materialized view victories.

2 Preliminaries

As our underlying language, we use the extension of Datalog that is proposed in [12]. We introduce tuples and sets as first-class citizens in our language. A tuple is an ordered sequence of Datalog constants or sets. A set is either empty or contains Datalog constants or tuples. For example, the tuple (yoda, vader, dagobah) and the sets $\{\}$, $\{$(yoda, vader, dagobah)$\}$ and $\{$(yoda, $\{1, 2\}$)$\}$ are legal in our language. We use the *setof* operator in our language to represent sets as follows.

Definition 1. Suppose $\phi(\bar{X}, \bar{Y})$ is a conjunction of subgoals. The setof subgoal setof($\bar{Y}$, $\phi(\bar{X}, \bar{Y})$, S) represents the set $S = \{\bar{Y} \mid \phi(\bar{X}, \bar{Y})\}$ for every binding of values in $\bar{X}$.

We illustrate the construction of sets in our language using the following example.

Example 1. Consider the running example that we presented in Section 1. Suppose that we would like to compute the set of characters who were defeated by Yoda at Dagobah. In our language, we compute the desired set using the following query.

$$q_1(S) \text{ :- } \text{setof}(D, \text{tournament}(\text{yoda}, D, \text{dagobah}), S)$$

In query q_1, the set $S = \{D \mid \text{tournament}(\text{yoda}, D, \text{dagobah})\}$. The evaluation of the query q_1 on the extension of tournament that is presented in Table 1 results in the answer tuple $q_1(\{\text{vader, palpatine}\})$.

We use the '|' operator to represent the decompositions of a set. The decomposition of a set S into an element $X \in S$ and the subset $S_1 = S \setminus \{X\}$ is represented as $\{X \mid S_1\}$. For example, $\{3 \mid \{1, 2\}\}$ represents the decomposition of the set of numbers $\{1, 2, 3\}$ into 3 and the subset $\{1, 2\}$. We define the predicate *member* in our language to check the membership of an element in a set. The member predicate has the signature member(X, S), where X is a Datalog constant or a tuple and S is a set. If $X \in S$ then member(X, S) is true, otherwise it is false. The member predicate can be defined in our language using the decomposition operator '|' operator as follows.

$$\text{member}(X, \{X \mid Y\})$$
$$\text{member}(Z, \{X \mid Y\}) \text{ :- } \text{member}(Z, Y)$$

In addition, we use $\cup$ and $\setminus$ operators in our language to represent set-union and set-difference respectively. We note that we can define $\cup$ and $\setminus$ operators in our language using the member predicate and the decomposition operator '|' although we do not define them as such in this paper.

Aggregation over sets: In our language, we define aggregates as predicates over sets. We define an aggregate either (a) in a stand-alone manner using the member predicate, the decomposition operator '|', and arithmetic operators or (b) as a view over other aggregates. For instance, we can compute the cardinality of a set in our language by inductively defining an aggregate, say count(X, C), as follows.

$$\begin{aligned}&\text{count}(\{\}, 0)\\&\text{count}(\{X \,|\, Y\}, C) \text{ :- } \text{count}(Y, C_1),\ C = C_1 + 1\end{aligned}$$

3 Maintenance of Aggregate Views

In the previous section, we discussed the specification of aggregates as predicates over sets in our language. Consider the running example that we presented in Section 1. Suppose that we would like to query the number of victories V achieved by a character W on a planet. We represent this query in our language as follows.

$$q(W, V) \text{ :- } \text{setof}(D, \text{tournament}(W, D, L), S),\ \text{count}(S, V)$$

For every binding of the variables W and L in the query q, V = cardinality of $\{D \mid \text{tournament}(W, D, L)\}$. In the Star Wars universe, this is equivalent to computing the number of victories V achieved by a character W on a planet. Since the answers to queries are computed under set semantics, the distinct numbers of victories are generated by the query q. To efficiently compute the answer to the query q, we can leverage the materialized view victories from our running example (in Section 1). The view victories(W, V) is defined as follows.

$$\text{victories}(W, V) \text{ :- } \text{setof}(D, \text{tournament}(W, D, L), S),\ \text{count}(S, V)$$

When changes are made to the edb relation tournament, we must maintain the materialized view victories to ensure the correctness of answers to the query q.

Maintenance of views that contain sets: Consider a view v in our language which is defined over the formula $\phi(\bar{X}, \bar{Y}, \bar{Z})$ using the aggregation predicate *agg* as follows.

$$v(\bar{X}, A) \text{ :- } \text{setof}(\bar{Y},\ \phi(\bar{X}, \bar{Y}, \bar{Z}),\ W),\ \text{agg}(W, A)$$

Since aggregates are defined as predicates over sets in our language, we can rewrite the definition of the view v using the following two rules, one of which contains a setof subgoal while the other does not.

$$\begin{aligned}v(\bar{X}, A) &\text{ :- } u(\bar{X}, W),\ \text{agg}(W, A)\\u(\bar{X}, W) &\text{ :- } \text{setof}(\bar{Y},\ \phi(\bar{X}, \bar{Y}, \bar{Z}),\ W)\end{aligned}$$

We note that if the definition of v contains k setof subgoals instead of one, we can rewrite the definition of v using $k + 1$ rules where only k of the rules contain setof subgoals. Since prior view maintenance techniques [7] already maintain views that do not contain sets, we focus *only* on the maintenance of views that contain setof subgoals. As a first step, we leverage differential relational calculus to incrementally propagate the changes in the edb relations to the views through *differential rules*. Then, in Section 4, we propose an algorithm called CReaM that applies these differential rules to optimally maintain materialized aggregate views.

Differential rules: In differential relational calculus, a database is represented as a set of edb relations and views $r_1, r_2, \ldots, r_k$ with arities $d_1, d_2, \ldots, d_k$. Each relation r_i is a set of d_i-tuples [5]. The changes to a relation r_i in the database consist of insertions of new tuples and deletions of existing tuples. The new state of a relation r_i after applying a change is represented as r_i'. An update to an existing tuple can be modeled as a deletion followed by an insertion. The insertion of new tuples into a relation r_i and the deletion of existing tuples from r_i are represented as the differential relations r_i^+ and r_i^- respectively. Prior work in [14] presents a set of *differential rules* to compute the differentials (v^+ or v^-) of a non-aggregate view v. We extend the framework that is presented in [14] to compute the differentials of aggregate views.

There are two possible ways in which a view v can be defined in our language using the setof operator over the formula $\phi(\bar{X}, \bar{Y}, \bar{Z})$.

1. The view v is defined as $v(\bar{X}, \bar{Z}, W)$:- setof($\bar{Y}$, $\phi(\bar{X}, \bar{Y}, \bar{Z})$, W). In this case, all of the variables of ϕ that are bound outside the setof subgoal are passed to the view v.

2. The view v is defined as $v(\bar{X}, W)$:- setof($\bar{Y}$, $\phi(\bar{X}, \bar{Y}, \bar{Z})$, W). In this case, not all of the variables of ϕ that are bound outside the setof subgoal are passed to the view v.

We present differential rules to compute the differentials of the view v in each of the above cases.

Case 1: Suppose that a view v is defined over a conjunction of subgoals $\phi(\bar{X}, \bar{Y})$ as follows.

$$v(\bar{X}, W) \text{ :- setof}(\bar{Y},\ \phi(\bar{X}, \bar{Y}),\ W)$$

Suppose that we define a view u over $v(\bar{X}, W)$ as $u(\bar{X})$:- $v(\bar{X}, W)$. In this case, we compute the differentials $v^+(\bar{X}, W)$ and $v^-(\bar{X}, W)$ as follows.

$$v^+(\bar{X}, W) \text{ :- setof}(\bar{Y},\ \phi^+(\bar{X}, \bar{Y}),\ W),\ \neg u(\bar{X}) \qquad (\Delta_1)$$
$$v^+(\bar{X}, W \cup W') \text{ :- setof}(\bar{Y},\ \phi^+(\bar{X}, \bar{Y}),\ W),\ v(\bar{X}, W') \qquad (\Delta_2)$$
$$v^+(\bar{X}, W' \setminus W) \text{ :- setof}(\bar{Y},\ \phi^-(\bar{X}, \bar{Y}),\ W),\ v(\bar{X}, W') \qquad (\Delta_3)$$
$$v^-(\bar{X}, W) \text{ :- setof}(\bar{Y},\ \phi^-(\bar{X}, \bar{Y}),\ W),\ v(\bar{X}, W) \qquad (\Delta_4)$$
$$v^-(\bar{X}, W) \text{ :- setof}(\bar{Y},\ \phi^+(\bar{X}, \bar{Y}),\ \_),\ v(\bar{X}, W) \qquad (\Delta_5)$$
$$v^-(\bar{X}, W) \text{ :- setof}(\bar{Y},\ \phi^-(\bar{X}, \bar{Y}),\ \_),\ v(\bar{X}, W) \qquad (\Delta_6)$$

In the above differential rules, '_' represents *don't care* variables.

Case 2: Now, suppose that a view v is defined over a conjunction of subgoals $\phi(\bar{X}, \bar{Y}, \bar{Z})$ as follows.

$$v(\bar{X}, W) \text{ :- setof}(\bar{Y},\ \phi(\bar{X}, \bar{Y}, \bar{Z}),\ W)$$

In addition, suppose that we define a view u over $v(\bar{X}, W)$ as $u(\bar{X})$:- $v(\bar{X}, W)$. In the definition of v, the set W is computed as $\{\bar{Y} \mid \phi(\bar{X}, \bar{Y}, \bar{Z})\}$ for every binding of $\bar{X}$ and $\bar{Z}$. Since $\bar{Z}$ is not passed to the view v, a view tuple, say $v(x, w)$, potentially has multiple derivations. In this case, we compute the differentials $v^+(\bar{X}, W)$ and $v^-(\bar{X}, W)$ as follows.

$$v^+(\bar{X}, W) \text{ :- setof}(\bar{Y},\ \phi^+(\bar{X}, \bar{Y}, \bar{Z}),\ W),\ \neg u(\bar{X}) \tag{Γ_1}$$

$$\begin{aligned} v^+(\bar{X}, W \cup W') \text{ :- } & \text{setof}(\bar{Y},\ \phi^+(\bar{X}, \bar{Y}, \bar{Z}),\ W),\ \text{setof}(\bar{Y}',\ \phi(\bar{X}, \bar{Y}', \bar{Z}),\ W'), \\ & \neg v(\bar{X}, W \cup W') \end{aligned} \tag{Γ_2}$$

$$\begin{aligned} v^+(\bar{X}, W' \setminus W) \text{ :- } & \text{setof}(\bar{Y},\ \phi^-(\bar{X}, \bar{Y}, \bar{Z}),\ W),\ \text{setof}(\bar{Y}',\ \phi(\bar{X}, \bar{Y}', \bar{Z}),\ W'), \\ & \neg v(X, W' \setminus W) \end{aligned} \tag{Γ_3}$$

$$\begin{aligned} v^-(\bar{X}, W) \text{ :- } & \text{setof}(\bar{Y},\ \phi^-(\bar{X}, \bar{Y}, \bar{Z}),\ W),\ v(\bar{X}, W), \\ & \neg\text{setof}(\bar{Y}',\ \phi'(\bar{X}, \bar{Y}', \bar{Z}),\ W) \end{aligned} \tag{Γ_4}$$

$$\begin{aligned} v^-(\bar{X}, W) \text{ :- } & \text{setof}(\bar{Y},\ \phi^+(\bar{X}, \bar{Y}, \bar{Z}),\ \_),\ \text{setof}(\bar{Y}',\ \phi(\bar{X}, \bar{Y}', \bar{Z}),\ W), \\ & \neg\text{setof}(\bar{Y}'',\ \phi'(\bar{X}, \bar{Y}'', \bar{Z}),\ W) \end{aligned} \tag{Γ_5}$$

$$\begin{aligned} v^-(\bar{X}, W) \text{ :- } & \text{setof}(\bar{Y},\ \phi^-(\bar{X}, \bar{Y}, \bar{Z}),\ \_),\ \text{setof}(\bar{Y}',\ \phi(\bar{X}, \bar{Y}', \bar{Z}),\ W), \\ & \neg\text{setof}(\bar{Y}'',\ \phi'(\bar{X}, \bar{Y}'', \bar{Z}),\ W) \end{aligned} \tag{Γ_6}$$

We note that we have omitted the correctness proofs of the proposed differential rules in this paper. However, the proofs can be found in the extended version of this paper [11].

4 Efficient Incremental Maintenance

In the previous section, we extended the differential rules that are presented in [14] to incrementally compute the differentials of views containing setof subgoals. In this section, we leverage the differential rules (from Section 3) to *optimally* maintain views containing setof subgoals. As a first step, we present an example where differential rules are leveraged to incrementally maintain aggregate views.

Example 2. Consider a materialized view *dominates* which is defined over the tournament relation from our running example (in Section 1) as follows.

$$\text{dominates}(V, W) \text{ :- setof}(D, \text{tournament}(V, D, L), W)$$

The extension of the view dominates that corresponds to the extension of tournament (in Table 1) is presented as follows.

dominates	
Victor	**Defeated**
yoda	{palpatine, vader}
vader	{yoda}
yoda	{palpatine}

Suppose that Yoda defeats Darth Vader at Tatooine in a new tournament match. This match results in the insertion of the tuple (yoda, vader, tatooine) into the tournament relation i.e. (yoda, vader, tatooine) $\in$ tournament$^+$. Since the non-aggregated variable L in tournament is not passed to the view dominates, we leverage the differential rules Γ_1– Γ_6 to incrementally compute the differentials of the view dominates. By applying the differential rules Γ_2 and Γ_5 on the differential tournament$^+$ and the relations tournament and dominates, we derive the differentials dominates$^-$(yoda, {palpatine}) and dominates$^+$(yoda, {palpatine, vader}). The computed differentials correspond to updating the tuple (yoda, {palpatine}) $\in$ dominates to the tuple (yoda, {palpatine, vader}).

We note that in Example 2, we access tournament's extension in addition to the differential tournament$^+$ to maintain the view dominates using differential rules. A tuple $(V, W) \in$ dominates could potentially have multiple derivations in tournament because V could defeat the *same* set of characters W at multiple planetary systems. Hence, additional accesses to the extensions of edb relations are required to maintain materialized views using differential rules.

Alternatively, we could maintain the count of the different derivations of tuples to optimize the maintenance of aggregate views. Prior technques [6, 8, 9, 13, 15] leverage this idea to optimize the maintenance of aggregate views where tuples in the view have multiple derivations in the edb relations. Suppose that in Example 2, we maintain the count of the different derivations of a tuple in a manner similar to the *counting algorithm* that is presented in [8].

dominates		
Victor	**Defeated**	**Number of Derivations**
yoda	{palpatine, vader}	1
vader	{yoda}	1
yoda	{palpatine}	1

Now suppose that we delete a tuple, say (yoda, vader, dagobah) from tournament's extension. In response to this deletion, we decrease the count of the tuple (yoda, {palpatine, vader}) $\in$ dominates from 1 to 0 (thereby deleting it from the view) and increase the count of the tuple (yoda, {palpatine}) from 1 to 2. The updated extension of the view dominates is presented below.

dominates		
Victor	**Defeated**	**Number of Derivations**
vader	{yoda}	1
yoda	{palpatine}	2

When the tuple (yoda, vader, dagobah) is deleted from tournament's extension, we do not have to access tournament's extension to incrementally maintain the materialized view dominates. However, consider a scenario where we delete the tuple (yoda, palpatine, dagobah) instead of the tuple (yoda, vader, palpatine) from tournament's extension. In this scenario, unless we access tournament's extension, we *cannot* correctly update the materialized view dominates because we do not have sufficient information to determine whether the existing tuple (yoda, {palpatine}) $\in$ dominates is to be deleted or the tuple (yoda, {palpatine, vader}) $\in$ dominates is to be updated.

Incremental Maintenance using CReaM[1]: Consider the materialized view dominates that we presented in Example 2. Suppose that we rewrite the definition of dominates using an auxiliary view v_a as follows.

$$\text{dominates}(V, W) \text{ :- } v_a(V, L, W)$$
$$v_a(V, L, W) \text{ :- setof}(D, \text{tournament}(V, D, L), W)$$

In addition, suppose that we materialize the auxiliary view v_a and maintain the counts of the derivations of a tuple in the view dominates. The extension of the auxiliary view v_a is presented as follows.

[1]The algorithm has been named CReaM because it **C**ounts the tuple derivations in a view, **Re**writes the view using auxiliary views and **M**aintains the auxiliary views.

v_a		
Victor	**Location**	**Defeated**
yoda	dagobah	{palpatine, vader}
vader	tatooine	{yoda}
yoda	tatooine	{palpatine}

Now, suppose that we delete the tuple (yoda, palpatine, dagobah) from the extension of tournament. Since all of the non-aggregated variables of tournament are passed to the auxiliary view v_a, we can incrementally maintain v_a using the differential rules Δ_1– Δ_6 (from Section 3). We note that Δ_1– Δ_6 only access the extension of a view and the differentials of the edb relations over which the view is defined. Thus, we are able to compute the differentials v_a^-(yoda, dagobah, {palpatine, vader}) and v_a^+(yoda, dagobah, {vader}) without accessing the extension of tournament.

Since the modified definition of the view dominates *does not* contain setof subgoals, we use the counting algorithm [8] to incrementally maintain the count of the tuple derivations in the view dominates in a subsequent step.

We now propose an algorithm called CReaM to incrementally maintain views that contain setof subgoals. The CReaM algorithm is presented in Figure 1.

Figure 1: Algorithm to *optimally* maintain views containing setof subgoals

```
**CReaM** Algorithm
Input:  1. Materialized view $v(\bar{X}, W)$ defined as:
              $v(\bar{X}, W)$ :- setof($\bar{Y}, \phi(\bar{X}, \bar{Y}, \bar{Z}), W$),
        2. Differentials $\phi^+(\bar{X}, \bar{Y}, \bar{Z})$ and $\phi^-(\bar{X}, \bar{Y}, \bar{Z})$
Step 1: Rewrite the view $v$ using an auxiliary view $v_a$ which contains all of the
        non-aggregated variables
            $v(\bar{X}, W)$ :- $v_a(\bar{X}, \bar{Z}, W)$
            $v_a(\bar{X}, \bar{Z}, W)$ :- setof($\bar{Y}, \phi(\bar{X}, \bar{Y}, \bar{Z}), W$)
Step 2: Materialize the auxiliary view $v_a$
Step 3: Maintain the count of the tuple derivations in the view $v$
Step 4: Apply the differential rules $\Delta_1$- $\Delta_6$ over $\phi^+(\bar{X}, \bar{Y}, \bar{Z})$ and $\phi^-(\bar{X}, \bar{Y}, \bar{Z})$
        to compute $v_a^+(\bar{X}, \bar{Z}, W)$ and $v_a^-(\bar{X}, \bar{Z}, W)$
        Use $v_a^+(\bar{X}, \bar{Z}, W)$ and $v_a^-(\bar{X}, \bar{Z}, W)$ to incrementally update the counts of $v$'s tuples
        using [9]
```

We note that CReaM incrementally maintains a view whose definition contains a single setof subgoal. However, when the supplied view definition contains multiple setof subgoals and aggregate predicates, we can incrementally maintain the view using CReaM as follows. Suppose that a materialized view v contains k setof subgoals $\{s_i\}$ and m aggregate predicates $\{a_i\}$. First, we rewrite the definition of v using k auxiliary predicates, say $\{t_i\}$ where each t_i is defined as t_i :- s_i. Next, we maintain the counts of the tuple derivations in v and incrementally compute the differentials of t_i by applying CReaM to the extensions of the auxiliary predicates $\{t_i\}$ and the differentials of the edb relations. Since the modified definition of v does not contain setof subgoals, we use the counting algorithm that is presented in [8] to incrementally maintain the materialized view v.

In the following theorem, we prove the correctness of the CReaM algorithm.

Theorem 1. *CReaM correctly maintains a materialized view containing setof subgoals.*

Proof. Consider a view v in our language which is defined using k setof subgoals $s_1, s_2, \ldots, s_k$ as v :- $s_1, s_2, \ldots, s_k$. Suppose that we introduce k auxiliary views $v_{a_1}, v_{a_2}, \ldots, v_{a_k}$ where each v_{a_i} is defined as v_{a_i} :- s_i. In the definition of the auxiliary view v_{a_i}, all of the variables that are bound outside the setof subgoal s_i are passed to the view. By replacing the setof subgoals using the auxiliary views, we can rewrite the definition of the view v as v :- $v_{a_1}, v_{a_2}, \ldots, v_{a_k}$. Since the modified definition of the view v does not contain setof subgoals, we can correctly maintain it by applying the counting algorithm [8]. In addition, we can use the result in [11] to prove that the differential rules Δ_1– Δ_6 correctly compute the differentials of the auxiliary views $\{v_{a_i}\}$. □

Next, we show that when the supplied materialized view and the auxiliary views that are materialized by CReaM are indexed, the time taken by CReaM to incrementally maintain a view is *optimal*.

Theorem 2. *CReaM optimally maintains a materialized view containing a setof subgoal in response to changes in edb relations when the physical design of the underlying database is optimized.*

Proof. Suppose that a materialized view v is supplied as an input to CReaM. In addition, suppose that v consists of n tuples. To maintain v, CReaM rewrites the definition of v using an auxiliary view (say v_a) and computes the differentials of v_a using Δ_1– Δ_6. When v and v_a are indexed, the time required to compute the differentials is $\mathrm{O}(\log n)$. The detailed analysis of the time complexity of CReaM is presented in [11]. Therefore, to prove the optimality of CReaM, it suffices to show that $\Omega(\log n)$ time is required to incrementally maintain a materialized view that contains n tuples.

To prove the lower bound, we reduce the problem of incrementally maintaining the *partial sums* of an array of n numbers to the problem of incrementally maintaining an extension of a view with n tuples. Prior work in [3, 4, 16] have independently proven that the maintenance of the partial sums of an array of n numbers requires $\Omega(\log n)$ time. Consider an array of n numbers $\{a_i\}$. The partial sums problem maintains the sum $\sum_{i=1}^{k} a_i$ for every k $(1 \leq k \leq n)$ subject to updates of the form $a_i = a_i + x$, where x is a number.

We reduce the instance of the partial sums problem over the array $\{a_i\}$ to an instance of the view maintenance problem as follows. Consider an instance of the view maintenance problem where we have two edb relations $r(A, B)$ and $s(B, C)$. The extension of $r(A, B)$ consists of the set of $n \times (n-1)$ tuples, $\{(i, j) \mid 1 \leq j \leq i \leq n\}$. The extension of $s(A, B)$ consists of the set of n tuples, $\{(i, a_i) \mid 1 \leq i \leq n\}$. Suppose that we materialize n views $v_1, v_2, \ldots, v_n$ over $r(A, B)$ and $s(B, C)$ where each v_i is defined as $v_i(S)$:- setof$((B, C), r(i, B) \,\&\, s(B, C), W)$, sum$(W, S, 2)$. In the definition of v_i, the aggregate sum$(W, S, 2)$ computes the sum of the 2nd component of the tuples $\in W$. When an array value a_i is updated to $a_i + x$, we update the tuple $(i, a_i) \in s(B, C)$ to the tuple$(i, a_i + x)$. Since we can compute the partial sum $\sum_{i=1}^{k} a_i$ by finding the value s which is in the extension of v_k, the problem of maintaining the partial sums of the array $\{a_i\}$ reduces to the problem of incrementally maintaining the views $v_1, v_2, \ldots, v_k$. Therefore, $\Omega(\log n)$ time is required to incrementally maintain a materialized view that contains n tuples. □

5 Related Work

The problem of incrementally maintaining views has been extensively studied in the database community. A survey of the view maintenance techniques is presented in [7]. The view maintenance algorithms proposed in [1,6,8–10,14,17,18] leverage differential relational algebra to incrementally maintain views in response to changes to the underlying edb relations. For instance [14] incrementally computes the differentials (or changes) of views by applying a set of differential rules over the extensions of edb relations and their differentials. However, only a small fraction of the prior work on incremental view maintenance [6, 8, 9, 13, 15, 18, 19] addresses the maintenance of aggregate views. The techniques proposed in [6, 8, 13, 15] incrementally maintain views having only one aggregation operator. Furthermore, the incremental maintenance algorithms presented in [6, 8, 9, 13, 15, 18, 19] can support only a fixed set of built-in aggregate operators (such as min, max, sum, and count).

Our work differs from prior work on incrementally maintaining aggregate views in two ways. First, we propose a view maintenance algorithm called CReaM that *optimally* maintains aggregate views. Second, we can extend the CReaM algorithm to maintain views that contain *user-defined* aggregates. To maintain views with user-defined aggregates, we rewrite the supplied view definitions using auxiliary views that contain setof subgoals and apply the CReaM algorithm to maintain the auxiliary views. Then, we apply prior maintenance algorithms [7] to maintain views whose definitions do not contain sets.

6 Conclusion

We propose an algorithm called CReaM that incrementally maintains materialized aggregate views in response to changes to edb relations by materializing auxiliary views and applying differential rules. When the physical design of the underlying database is optimized, CReaM optimally maintains the supplied aggregate views.

References

[1] Jose A. Blakeley, Per-Ake Larson, and Frank W. Tompa. Efficiently updating materialized views. In *SIGMOD*, 1986.

[2] Latha S. Colby, Akira Kawaguchi, Daniel F. Lieuwen, Inderpal S. Mumick, and Kenneth A. Ross. Supporting multiple view maintenance policies. In *SIGMOD*, 1997.

[3] Michael L. Fredman. A lower bound on the complexity of orthogonal range queries. *J. ACM*, 1981.

[4] Michael L. Fredman. The complexity of maintaining an array and computing its partial sums. *J. ACM*, 1982.

[5] Herve Gallaire, Jack Minker, and Jean-Marie Nicolas. Logic and databases: A deductive approach. *ACM Computing Surveys*, 1984.

[6] Timothy Griffin and Leonid Libkin. Incremental maintenance of views with duplicates. In *SIGMOD*, 1995.

[7] Ashish Gupta and Inderpal S. Mumick. *Materialized Views:Techniques, Implementations, and Applications*. MIT Press, 1999.

[8] Ashish Gupta, Inderpal S. Mumick, and V. S. Subrahmanian. Maintaining views incrementally. In *SIGMOD*, 1993.

[9] Himanshu Gupta and Inderpal S. Mumick. Incremental maintenance of aggregate and outerjoin expressions. *Information Systems*, 2006.

[10] Volker Kuchenhoff. On the efficient computation of the difference between consecutive database states. *DOOD*, 1991.

[11] Abhijeet Mohapatra and Michael Genesereth. Incremental maintenance of aggregate views. Technical Report LG-2013-01, Stanford University, 2013. `http://logic.stanford.edu/reports/LG-2013-01.pdf`.

[12] Abhijeet Mohapatra and Michael Genesereth. Reformulating aggregate queries using views. In *SARA*, 2013.

[13] Inderpal S. Mumick, Dallan Quass, and Barinderpal S. Mumick. Maintenance of data cubes and summary tables in a warehouse. In *SIGMOD*, 1997.

[14] Levent V. Orman. Differential relational calculus for integrity maintenance. *ACM TKDE*, 1998.

[15] Themistoklis Palpanas, Richard Sidle, Roberta Cochrane, and Hamid Pirahesh. Incremental maintenance for non-distributive aggregate functions. In *VLDB*, 2002.

[16] Mihai Păatraşcu and Erik D. Demaine. Tight bounds for the partial-sums problem. In *SODA*, 2004.

[17] Xiaolei Qian and Gio Wiederhold. Incremental recomputation of active relational expressions. *ACM TKDE*, 1991.

[18] Dallan Quass. Maintenance expressions for views with aggregation. In *Views*, 1996.

[19] Dallan Quass. *Materialized views in data warehouses*. PhD thesis, Stanford University, 1998.