Algebraic Change Propagation for Semijoin and Outerjoin Queries

Timothy Griffin, Bharat Kumar
Bell Laboratories
Lucent Technologies
{griffin,bharat}@research.bell-labs.com

Abstract

Many interesting examples in view maintenance involve semijoin and outerjoin queries. In this paper we develop algebraic change propagation algorithms for the following operators: semijoin, anti-semijoin, left outerjoin, right outerjoin, and full outerjoin.

1 Motivation

View maintenance algorithms are important for data warehouses, database integration, and efficient checking of integrity constraints [BLT86, CGL+96, GL95, GLT97, GJM97, GMS93, HZ96, QW91, RSS96, ZGHW95, QGMW96]. Many practical examples of view maintenance involve semijoin and outerjoin queries. Algorithms for incremental maintenance of such queries was presented by Gupta et. al in [GJM97]. That paper presented procedural algorithms, but left open as a challenge their algebraic formulation in the style of Griffin et. al [GL95, GLT97].

In this paper we develop algebraic change propagation algorithms for the following operators: semijoin, anti-semijoin, left outerjoin, right outerjoin, and full outerjoin. There are several advantages to an algebraic formulation, over a procedural approach, of incremental view maintenance algorithms. First, the results are amenable to compile-time and run-time optimizations. Second, the correctness of algebraic algorithms is usually more apparent. Finally, two algebraic algorithms for two distinct relational operators can easily be composed, using the approach of [GL95, GLT97], to provide an algorithm for queries involving both operators. This is not always the case with procedural algorithms.

Outline. Section 2 reviews change propagation rules (under single update) from [QW91, GL95, GLT97], and extends them for the semijoin and outerjoin operators. Section 3 reviews change propagation queries (under multiple updates) for the base operators, and presents corresponding propagation queries for the derived operators.

2 Single Update Propagation

Let R, R_1, R_2, ... denote names of relations in a database scheme. Let p range over quantifier-free predicates, and A range over sets of attribute names. Queries are generated by the grammar

S	$::=$	R	base relation
	$\mid$	$\sigma_p(S)$	selection
	$\mid$	$\Pi_A(S)$	projection
	$\mid$	$S \times S$	cartesian product
	$\mid$	$S \cup S$	union
	$\mid$	$S \cap S$	intersection
	$\mid$	$S - S$	difference
	$\mid$	$S \bowtie S$	natural join

The symbols Q, S, T will be used to denote arbitrary queries. The notation $T = S$ means that for all database states, the queries T and S evaluate to the same set of tuples. Similarly, the notation $T \subseteq S$ means that for all database states, the result of evaluating query T is a subset of the result of evaluating query S.

We can define *derived* operators such as semijoin and outerjoin operators in terms of the above. Notational definitions for these operators are presented in Figure 1. The notation $[S]$ denotes the set of attributes of the query S. The tuple d_Q represents the *default* tuple of query Q. In SQL, this would be a tuple of the correct arity, with only NULL values.

$$
\begin{array}{rcll}
S \ltimes T & = & \Pi_{[S]}(S \bowtie T) & \text{semijoin (D1)} \\
S \overline{\ltimes} T & = & S - (S \ltimes T) & \text{anti-semijoin (D2)} \\
S \sqsupset\!\!\bowtie T & = & (S \bowtie T) \cup ((S \overline{\ltimes} T) \times \{d_T\}) & \text{left outerjoin (D3)} \\
S \bowtie\!\!\sqsubset T & = & (S \bowtie T) \cup (\{d_S\} \times (T \overline{\ltimes} S)) & \text{right outerjoin (D4)} \\
S \sqsupset\!\!\bowtie\!\!\sqsubset T & = & (S \sqsupset\!\!\bowtie T) \cup (\{d_S\} \times (T \overline{\ltimes} S)) & \text{full outerjoin (D5)}
\end{array}
$$

Figure 1: Derived operators.

For ease of presentation, we only consider equality join predicates. We further assume that both the queries participating in the join operations have the same attribute names on which that join is performed. For example, in our notation,

$$R \ltimes_A S = R \ltimes_{(R.A = S.A)} S$$

In addition, we shall drop the attribute names where unnecessary, hence the above would be written as $R \ltimes S$.

2.1 Basic Propagation Rules

Figure 2 is taken from [QW91] and represents the basic collection of change propagation equations (for single changes) on which all of our work depends. The notation ∇S and $\triangle S$ represents the delete and insert sets respectively for query S. When read from left to right, these equations tell us how to propagate an insert/delete through a relational operator.

$$
\begin{array}{rl}
P1. & \sigma_p(S - \nabla S) = \sigma_p(S) - \sigma_p(\nabla S) \\
P2. & \sigma_p(S \cup \triangle S) = \sigma_p(S) \cup \sigma_p(\triangle S) \\
P3. & \Pi_A(S - \nabla S) = \Pi_A(S) - (\Pi_A(S) - \Pi_A(S - \nabla S)) \\
P4. & \Pi_A(S \cup \triangle S) = \Pi_A(S) \cup \Pi_A(\triangle S) \\
P5. & (S - \nabla S) \times T = (S \times T) - (\nabla S \times T) \\
P6. & (S \cup \triangle S) \times T = (S \times T) \cup (\triangle S \times T) \\
P7. & (S - \nabla S) \cup T = (S \cup T) - (\nabla S - T) \\
P8. & (S \cup \triangle S) \cup T = (S \cup T) \cup (\triangle S - T) \\
P9. & (S - \nabla S) - T = (S - T) - (\nabla S - T) \\
P10. & S - (T - \nabla T) = (S - T) \cup (S \cap \nabla T) \\
P11. & (S \cup \triangle S) - T = (S - T) \cup (\triangle S - T) \\
P12. & S - (T \cup \triangle T) = (S - T) - (S \cap \triangle T) \\
P13. & (S - \nabla S) \cap T = (S \cap T) - (\nabla S \cap T) \\
P14. & (S \cup \triangle S) \cap T = (S \cap T) \cup (\triangle S \cap T) \\
P15. & (S - \nabla S) \bowtie T = (S \bowtie T) - (\nabla S \bowtie T) \\
P16. & (S \cup \triangle S) \bowtie T = (S \bowtie T) \cup (\triangle S \bowtie T)
\end{array}
$$

Figure 2: Relational equations used in change propagation for base operators.

Let us consider using the propagation equations of Figure 2 to derive a propagation equation for the semijoin operator. Suppose we have a view V that contains the result of query $S \ltimes T$. Assume that ∇T is deleted from T. We want to compute the change to V, i.e., we want find a query X that solves the equation:

$$S \ltimes (T - \nabla T) = (S \ltimes T) - X$$

Since semijoin is a derived operator, we should be able to expand its definition and apply the propagation rules of Figure 2. It will be useful to have two auxiliary rules. The first is alternative to P3 under the assumption that $\nabla S \subseteq S$:

$$P3'. \quad \Pi_A(S - \nabla S) = \Pi_A(S) - (\Pi_A(\nabla S) - \Pi_A(S - \nabla S))$$

We also make use of the following equality:

$$\text{R1.} \quad (Q \ltimes S) - (Q \ltimes T) = Q \ltimes (S \overline{\ltimes} T)$$

With these equations, we can now derive a concise solution to the semijoin equation above (note that we are not assuming that $\nabla T \subseteq T$):

$$
\begin{array}{rl}
 & S \ltimes (T - \nabla T) \\
= & \Pi(S \bowtie (T - \nabla T)) \quad \text{(by definition)} \\
= & \Pi(S \bowtie (T - (\nabla T \cap T))) \\
= & \Pi((S \bowtie T) - (S \bowtie (\nabla T \cap T))) \quad \text{(by P15)} \\
= & \Pi(S \bowtie T) - \\
 & [\Pi(S \bowtie (\nabla T \cap T)) - \Pi(S \bowtie (T - \nabla T))] \quad \text{(by } P3'\text{)} \\
= & (S \ltimes T) - [(S \ltimes (\nabla T \cap T)) - (S \ltimes (T - \nabla T))] \\
= & (S \ltimes T) - (S \ltimes ((\nabla T \cap T) \overline{\ltimes} (T - \nabla T))) \quad \text{(by R1)}
\end{array}
$$

Therefore, our solution is

$$X = (S \ltimes ((\nabla T \cap T) \overline{\ltimes} (T - \nabla T)))$$

A word of explanation is in order. Let us say that for tuples $t_1, t_2 \in T$, t_1 is a *friend* of t_2 under predicate P iff $\{t_1\} \bowtie_P \{t_2\} \neq \phi$. Now, the sub-expression

$$(\nabla T \cap T) \overline{\ltimes} (T - \nabla T)$$

represents the subset of tuples actually deleted from T that do not leave any friends behind. We will call such tuples *loyal*. If a tuple $s \in S$ joins with any of of the

$$
\begin{array}{ll}
P17. & (S - \bigtriangledown S) \ltimes T = (S \ltimes T) - \bigtriangledown S \\
P18. & (S \cup \bigtriangleup S) \ltimes T = (S \ltimes T) \cup (\bigtriangleup S \ltimes T) \\
P19. & S \ltimes (T - \bigtriangledown T) = (S \ltimes T) - (S \ltimes ((\bigtriangledown T \cap T) \overline{\ltimes} (T - \bigtriangledown T))) \\
P20. & S \ltimes (T \cup \bigtriangleup T) = (S \ltimes T) \cup (S \ltimes \bigtriangleup T) \\
 & \\
P21. & (S - \bigtriangledown S) \overline{\ltimes} T = (S \overline{\ltimes} T) - \bigtriangledown S \\
P22. & (S \cup \bigtriangleup S) \overline{\ltimes} T = (S \overline{\ltimes} T) \cup (\bigtriangleup S \overline{\ltimes} T) \\
P23. & S \overline{\ltimes} (T - \bigtriangledown T) = (S \overline{\ltimes} T) \cup (S \ltimes ((\bigtriangledown T \cap T) \overline{\ltimes} (T - \bigtriangledown T))) \\
P24. & S \overline{\ltimes} (T \cup \bigtriangleup T) = (S \overline{\ltimes} T) - (S \ltimes \bigtriangleup T) \\
 & \\
P25. & (S - \bigtriangledown S) \sqsupset\!\!\bowtie T = (S \sqsupset\!\!\bowtie T) - (\bigtriangledown S \sqsupset\!\!\bowtie T) \\
P26. & (S \cup \bigtriangleup S) \sqsupset\!\!\bowtie T = (S \sqsupset\!\!\bowtie T) \cup (\bigtriangleup S \sqsupset\!\!\bowtie T) \\
P27. & S \sqsupset\!\!\bowtie (T - \bigtriangledown T) = ((S \sqsupset\!\!\bowtie T) - (S \bowtie \bigtriangledown T)) \cup ((S \ltimes ((T \cap \bigtriangledown T) \overline{\ltimes} (T - \bigtriangledown T))) \times \{d_T\}) \\
P28. & S \sqsupset\!\!\bowtie (T \cup \bigtriangleup T) = ((S \sqsupset\!\!\bowtie T) - ((S \ltimes \bigtriangleup T) \times \{d_T\})) \cup (S \bowtie \bigtriangleup T) \\
 & \\
P29. & (S - \bigtriangledown S) \bowtie\!\!\sqsubset T = ((S \bowtie\!\!\sqsubset T) - (\bigtriangledown S \bowtie T)) \cup (\{d_S\} \times (T \ltimes ((S \cap \bigtriangledown S) \overline{\ltimes} (S - \bigtriangledown S)))) \\
P30. & (S \cup \bigtriangleup S) \bowtie\!\!\sqsubset T = ((S \bowtie\!\!\sqsubset T) - (\{d_S\} \times (T \ltimes \bigtriangleup S))) \cup (\bigtriangleup S \bowtie T) \\
P31. & S \bowtie\!\!\sqsubset (T - \bigtriangledown T) = (S \bowtie\!\!\sqsubset T) - (S \bowtie\!\!\sqsubset \bigtriangledown T) \\
P32. & S \bowtie\!\!\sqsubset (T \cup \bigtriangleup T) = (S \bowtie\!\!\sqsubset T) \cup (S \bowtie\!\!\sqsubset \bigtriangleup T) \\
 & \\
P33. & (S - \bigtriangledown S) \sqsupset\!\!\bowtie\!\!\sqsubset T = (S \sqsupset\!\!\bowtie\!\!\sqsubset T) - (\bigtriangledown S \sqsupset\!\!\bowtie T) \cup (\{d_S\} \times (T \ltimes ((\bigtriangledown S \cap S) \overline{\ltimes} (S - \bigtriangledown S)))) \\
P34. & (S \cup \bigtriangleup S) \sqsupset\!\!\bowtie\!\!\sqsubset T = (S \sqsupset\!\!\bowtie\!\!\sqsubset T) \cup (\bigtriangleup S \sqsupset\!\!\bowtie T) - (\{d_S\} \times (T \ltimes \bigtriangleup S)) \\
P35. & S \sqsupset\!\!\bowtie\!\!\sqsubset (T - \bigtriangledown T) = (S \sqsupset\!\!\bowtie\!\!\sqsubset T) - (S \bowtie\!\!\sqsubset \bigtriangledown T) \cup ((S \ltimes ((\bigtriangledown T \cap T) \overline{\ltimes} (T - \bigtriangledown T))) \times \{d_T\}) \\
P36. & S \sqsupset\!\!\bowtie\!\!\sqsubset (T \cup \bigtriangleup T) = (S \sqsupset\!\!\bowtie\!\!\sqsubset T) \cup (S \bowtie\!\!\sqsubset \bigtriangleup T) - ((S \ltimes \bigtriangleup T) \times \{d_T\})
\end{array}
$$

Figure 3: Relational equations used in change propagation for semijoins and outerjoins.

loyal tuples, then it will have to be deleted from the view $S \ltimes T$ since there will be no tuples left in T that join with s.

The other change propagation rules for semijoin and outerjoin operators can be derived in a similar way. The complete set of propagation rules for these operators is listed in Figure 3. Note that equations P19, P23, P27, P29, P33 and P35 contain sub-expressions for loyal tuples. Also note that, unlike Figure 2, the right hand side of some equations in Figure 3 contain both delete and insert queries. This is due to the fact that the expansion of the left hand side of these equations will result in a query where the change set occurs both negatively and positively (using the terminology of [QW91]). For example, in the expansion

$$
\begin{array}{ll}
 & S \sqsupset\!\!\bowtie (T \cup \bigtriangleup T) \\
= & (S \bowtie (T \cup \bigtriangleup T)) \cup (S - (S \ltimes (T \cup \bigtriangleup T))) \times \{d_T\}
\end{array}
$$

the subexpression $\bigtriangleup T$ appears both negatively and positively.

We shall use the following example to illustrate the results obtained in this paper.

EXAMPLE 2.1 CourseReg

Consider two relations `Reg` and `Course` that hold the set of students along with the courses they have registered for, and the set of courses offered by a university, respectively. The schemas of the two relations are defined as (for ease of presentation, we have kept the schemas very simple):

`Reg` : [`student, cname`]
`Course` : [`course, iname`]

where `cname` is the name of the course taken by a student, and `iname` is the name of the instructor teaching a particular course. We allow that students can be instructors too. Let the initial set of tuples in these relations be:

Reg

student	cname
Joe	DB
Mary	DB
Sam	Psych

Course

course	iname
DB	Bob
AI	Tom
History	Jack
Psych	Jill

Let us define a view `CourseReg` which contains the set of all the courses offered along with the set of students (if any) taking those courses, i.e.,

$$\texttt{CourseReg} = \texttt{Course} \supset\!\!\bowtie_{\texttt{course}=\texttt{cname}} \texttt{Reg}$$

The view `CourseReg` will have the following tuples (to keep the propagation rules simple for illustration later on in the section, we leave both the duplicate columns `course` and `cname` in the view):

CourseReg			
`course`	`iname`	`student`	`cname`
AI	Tom	NULL	NULL
DB	Bob	Joe	DB
DB	Bob	Mary	DB
History	Jack	NULL	NULL
Psych	Jill	Sam	Psych

Suppose the following insert is made to the `Reg` table:

$$\triangle\texttt{Reg} = \{< \text{Jill, AI} >\}$$

From P28 in Figure 3 we see that the effect of this insert on `CourseReg` will be to perform the following updates to the view (the join predicate is implicit here):

$$\begin{aligned}
\triangledown\texttt{CourseReg} &= (\texttt{Course} \triangleright \{< \text{Jill, AI} >\}) \times \\
&\quad \{< \text{NULL, NULL} >\} \\
&= \{<\text{AI, Tom, NULL, NULL}>\}, \\
\triangle\texttt{CourseReg} &= \texttt{Course} \bowtie \{< \text{Jill, AI} >\} \\
&= \{<\text{AI, Tom, Jill, AI}>\}
\end{aligned}$$

The final contents of `CourseReg` will be:

CourseReg (updated)			
`course`	`iname`	`student`	`cname`
AI	Tom	Jill	AI
DB	Bob	Joe	DB
DB	Bob	Mary	DB
History	Jack	NULL	NULL
Psych	Jill	Sam	Psych

3 Multi-Update Propagation

The last section presented simple propagation rules that treat only single updates to a single relation. In this section, we consider simultaneous updates to multiple relations. In order to formalize this, we introduce the concept of a *change substitution*. Change substitutions are substitutions of the form

$$\begin{aligned}
\eta \;=\; & \{R_1 \leftarrow (R_1 - \triangledown R_1) \cup \triangle R_1, \ldots, \\
& R_n \leftarrow (R_n - \triangledown R_n) \cup \triangle R_n\},
\end{aligned}$$

where the expressions $\triangledown R_i$ and $\triangle R_i$ are queries that represent sets deleted from and inserted into base relation R_i respectively. If Q is a query, then $\eta(Q)$ represents the application of the substitution η to Q, and represents the query constructed from Q by replacing each occurrence of table name R_i by the query $(R_1 - \triangledown R_1) \cup \triangle R_1$.

For incremental view maintenance, the substitution η can either be derived from an update transaction or from a collection of change logs. In the first case, $\eta(Q)$ represents the value that Q will have in after the update transaction has committed. In the case of change logs, $\eta(Q)$ represents the value that Q had at a point in the past when the change logs were initialized. For a detailed discussion of "past" and "future" queries, see [CGL+96].

Suppose Q is a query and η is a change substitution. We would like to determine how η's changes to the base relations propagate to changes in the value of Q. There are several ways of computing changes to Q. One approach is to iteratively apply the propagation rules to individual updates in any order we like. In general, this would result in a solution of the form

$$\eta(Q) = (\cdots((Q \text{ op}_1\ E_1) \text{ op}_2\ E_2) \ \cdots \ \text{op}_{2n}\ E_{2n})$$

where $\text{op}_i \in \{-, \cup\}$ and E_i is obtained from "pulling up" the delete or insert set of some relation by repeatedly applying simple propagation rules. Suppose we want a result of the form:

$$\eta(Q) = (Q - \triangledown Q) \cup \triangle Q. \tag{1}$$

For the simple operators, Qian and Wiederhold [QW91] showed that this can be accomplished simply by applying the propagation rules in the correct order. However, this approach will not work for many of the propagation rules in Figure 3 since a single insert or delete can propagate to an insert *and* a delete query.

Alternatively, we can follow the approach of [GL95, GLT97] and define recursive functions $\triangle(\eta, Q)$ and $\triangledown(\eta, Q)$, such that

$$\eta(Q) = (Q - \triangledown(\eta, Q)) \cup \triangle(\eta, Q). \tag{2}$$

This approach naturally extends to propagation rules for the derived semijoin and outerjoin operators. The resulting queries are greatly simplified by assuming that the change substitution contain only *minimal* updates. Following [QW91], we define minimality as follows:

Q	∇Q
$S \ltimes T$	$(S \ltimes (\nabla T \overline{\ltimes} T^m)) \cup (\nabla S \ltimes (T - \nabla T)) \cup (\nabla S \ltimes (\Delta T \ltimes T))$
$S \overline{\ltimes} T$	$((S - \nabla S) \ltimes (\Delta T \overline{\ltimes} T)) \cup (\nabla S \overline{\ltimes} T)$
$S \sqsupset\!\!\bowtie T$	$(\nabla S \sqsupset\!\!\bowtie T) \cup (S \bowtie \nabla T) \cup (((S - \nabla S) \ltimes (\Delta T \overline{\ltimes} T)) \times \{d_T\})$
$S \bowtie\!\!\sqsubset T$	$(S \bowtie\!\!\sqsubset \nabla T) \cup (\nabla S \bowtie T) \cup (\{d_S\} \times ((T - \nabla T) \ltimes (\Delta S \overline{\ltimes} S)))$
$S \sqsupset\!\!\bowtie\!\!\sqsubset T$	$(\nabla S \sqsupset\!\!\bowtie T) \cup (S \bowtie\!\!\sqsubset \nabla T) \cup (((S - \nabla S) \ltimes (\Delta T \overline{\ltimes} T)) \times \{d_T\}) \cup (\{d_S\} \times ((T - \nabla T) \ltimes (\Delta S \overline{\ltimes} S)))$

Q	ΔQ
$S \ltimes T$	$((S - \nabla S) \ltimes (\Delta T \overline{\ltimes} T)) \cup (\Delta S \ltimes T^m)$
$S \overline{\ltimes} T$	$((S - \nabla S) \ltimes (\nabla T \overline{\ltimes} T^m)) \cup (\Delta S \overline{\ltimes} T^m)$
$S \sqsupset\!\!\bowtie T$	$(\Delta S \sqsupset\!\!\bowtie T^m) \cup (S^m \bowtie \Delta T) \cup (((S - \nabla S) \ltimes (\nabla T \overline{\ltimes} T^m)) \times \{d_T\})$
$S \bowtie\!\!\sqsubset T$	$(S^m \bowtie\!\!\sqsubset \Delta T) \cup (\Delta S \bowtie T^m) \cup (\{d_S\} \times ((T - \nabla T) \ltimes (\nabla S \overline{\ltimes} S^m)))$
$S \sqsupset\!\!\bowtie\!\!\sqsubset T$	$(\Delta S \sqsupset\!\!\bowtie T^m) \cup (S^m \bowtie\!\!\sqsubset \Delta T) \cup (((S - \nabla S) \ltimes (\nabla T \overline{\ltimes} T^m)) \times \{d_T\}) \cup (\{d_S\} \times ((T - \nabla T) \ltimes (\nabla S \overline{\ltimes} S^m)))$

Figure 4: Mutually recursive functions ∇ and Δ.

(a) $\nabla(\eta, Q) \subseteq Q$: All deleted tuples are in the set generated by Q.

(b) $Q \cap \Delta(\eta, Q) = \phi$: All inserted tuples are new.

We also assume that each change substitution η is minimal in the sense that for each i we have: 1) $\nabla R_i \subseteq R_i$, and 2) $R_i \cap \Delta R_i = \phi$.

For simplicity, we have abbreviated $\Delta(\eta, Q)$ as ΔQ and $\nabla(\eta, Q)$ as ∇Q. In this figure, and throughout this paper, the notation Q^m represents any query equivalent to $\eta(Q)$.

Based on the propagation rules listed in Figure 3, minimal change propagation queries for semijoin and outerjoin operators can be derived, and are listed in Figure 4. It is important to mention here that these queries are not unique, and it is possible to derive different equivalent forms.

EXAMPLE 3.1 `FacultyCourseReg`

Continuing with the example presented in the previous section, consider another view `FacultyCourseReg` which is defined as a subset of `CourseReg` where the instructor teaching the course is not a student. In relational form:

$$\texttt{FacultyCourseReg} = \texttt{CourseReg} \; \overline{\ltimes}_{\texttt{iname}=\texttt{Reg.student}} \texttt{Reg}$$

The initial value of this view (before the insert into `Reg`) is:

FacultyCourseReg			
`course`	`iname`	`student`	`cname`
AI	Tom	NULL	NULL
DB	Bob	Joe	DB
DB	Bob	Mary	DB
History	Jack	NULL	NULL
Psych	Jill	Sam	Psych

Let us determine the updates to this view on the same insert to `Reg` as in Example 2.1. From Figure 4 (after some simplication since we know that $\Delta\texttt{Course} = \nabla\texttt{Course} = \nabla\texttt{Reg} = \phi$), we see that (note that the anti-semijoin predicate is now different than the first example):

$$\begin{aligned}
&\nabla\texttt{FacultyCourseReg} \\
&= (\texttt{CourseReg} - \nabla\texttt{CourseReg}) \ltimes (\Delta\texttt{Reg} \overline{\ltimes} \texttt{Reg}) \cup \\
&\quad \nabla\texttt{CourseReg} \overline{\ltimes} \texttt{Reg} \\
&= (\texttt{CourseReg} - \{< \text{AI, Tom, NULL, NULL} >\}) \ltimes \\
&\quad (\{< \text{Jill, AI} >\} \overline{\ltimes} \texttt{Reg}) \cup \\
&\quad \{< \text{AI, Tom, NULL, NULL} >\} \overline{\ltimes} \texttt{Reg} \\
&= \{<\text{AI, Tom, NULL, NULL}>\} \cup \\
&\quad \{<\text{Psych, Jill, Sam, Psych}>\},
\end{aligned}$$

and,

$$
\begin{aligned}
&\triangle\texttt{FacultyCourseReg}\\
&= \triangle\texttt{CourseReg}\,\overline{\bowtie}\,(\texttt{Reg} \cup \triangle\texttt{Reg})\\
&= \{<\text{AI, Tom, Jill, AI}>\}\,\overline{\bowtie}\,(\texttt{Reg} \cup \{<\text{Jill, AI}>\})\\
&= \{<\text{AI, Tom, Jill, AI}>\}
\end{aligned}
$$

The final result of the view will be:

`FacultyCourseReg` (updated)			
`course`	`iname`	`student`	`cname`
AI	Tom	Jill	AI
DB	Bob	Joe	DB
DB	Bob	Mary	DB
History	Jack	NULL	NULL

4 Acknowledgments

We would like to thank Daniel Lieuwen for his comments on improving the presentation of the paper.

References

[BLT86] J.A. Blakeley, P.-A. Larson, and F.W. Tompa. Efficiently updating materialized views. In *SIGMOD*, pages 61–71, 1986.

[CGL+96] L. Colby, T. Griffin, L. Libkin, I. S. Mumick, and H. Trickey. Algorithms for deferred view maintenance. In *SIGMOD*, pages 469–480. ACM Press, 1996.

[GJM97] A. Gupta, H.V. Jagadish, and I.S. Mumick. Maintenance and self-maintenance of outerjoin views. In *Next Generation Information Technology and Systems*, 1997.

[GL95] T. Griffin and L. Libkin. Incremental maintenance of views with duplicates. In *SIGMOD*, pages 328–339. ACM Press, 1995.

[GLT97] Timothy Griffin, Leonid Libkin, and Howard Trickey. An improved algorithm for incremental recomputation of active relational expressions. *IEEE Transactions on Knowledge and Data Engineering*, 9(3):508–511, 1997.

[GMS93] A. Gupta, I.S. Mumick, and V.S. Subrahmanian. Maintaining views incrementally. In *SIGMOD*, pages 157–166, 1993.

[HZ96] R. Hull and G. Zhou. A framework for supporting data integration using the materialized and virtual approaches. In *SIGMOD*, pages 481–492, 1996.

[QGMW96] D. Quass, A. Gupta, I.S. Mumick, and J. Widom. Making views self-maintainable for data warehousing. In *Parallel and Distributed Information Systems*, pages 1–30, 1996.

[QW91] X. Qian and G. Wiederhold. Incremental recomputation of active relational expressions. *IEEE Transactions on Knowledge and Data Engineering*, 3(3):337–341, 1991.

[RSS96] K.A. Ross, D. Srivastava, and S. Sudarshan. Materialized view maintenance and integrity constraint checking: Trading space for time. In *SIGMOD*, pages 447–458. ACM Press, June 1996.

[ZGHW95] Y. Zhuge, H. Garcia-Molina, J. Hammer, and J. Widom. View maintenance in a warehousing environment. In *SIGMOD*, pages 316–327, San Jose, California, May 1995.