Acta Cybernetica **16** (2003) 93–131.

Derivation of Incremental Equations for PNF Nested Relations

Jixue Liu* and Millist Vincent*

Abstract

Incremental view maintenance techniques are required for many new types of data models that are being increasingly used in industry. One of these models is the nested relational model that is used in the modelling complex objects in databases. In this paper we derive a group of expressions for incrementally evaluating query expressions in the nested relational model. We also present an algorithm to propagate base relation updates to a materialized view when the view is defined as a complex query.

Keywords: view maintenance, data warehousing, nested databases, partitioned normal form, incremental computation.

1 Introduction

Materialized views are stored data collections that are derived from source data. Materialized views have attracted a significant amount of attention in recent years because of their importance in data warehousing [5, 7, 20]. In using materialized views, an issue of fundamental significance is developing efficient methods for updating the materialized views in response to changes in the source data; a procedure referred to as *view maintenance*. To maintain a materialized view, one has in general a choice between recomputing the view from scratch or maintaining the views incrementally. The incremental method is generally considered to be less expensive [13, 4, 6] since the size of an update to the source data is generally small in relation to the size of the source data. To maintain a view incrementally, one computes the new view using the updates to the source data, the old view and possibly some source data. For example, let the view V be defined in the flat relational model (using set semantics) as $V = R_1 \bowtie R_2$. For an insertion δR_1 to R_1, the incremental technique calculates the change to V as $\delta V = \delta R_1 \bowtie R_2$ and computes the new view, V^{new}, by $V^{new} = V^{old} \cup \delta V$ (where V^{old} equals $R_1 \bowtie R_2$) [13, 6]. This expression is called an *incremental propagation expression* (or *incremental expression* (IE) for short) for the Join operator.

*School of Computer and Information Science, University of South Australia, Mawson Lakes, SA5095, Australia. Email: {j.liu, vincent}@cs.unisa.edu.au

Incremental expressions for updating materialized views depend on the data model and query operators. Up to now, incremental equations have been derived for the models of flat relations [13], bags [4], and temporal data models [21]. Incremental equations for the nested relational model, on the other hand, have not been studied. The nested relational model is important because of its usage in modelling complex objects, a feature that has been incorporated in several commercial database systems such as Oracle8 and Illustra [18]. The nested model has also been used in data warehouses to model complex semantics [3], where incremental view maintenance has critical impact on system performances [20]. Further, the nested relational model is an important subclass of the object- relational model; a model that has been predicted to become the industry standard within the next few years [18]. Motivated by these observations, in this paper we derive IEs and develop a view maintenance algorithm for the nested relational model.

Several variations of the nested relational model have been proposed in the literature, depending on whether null values are permitted [10], whether empty sets are permitted [2], whether atomic attributes form a key and what data manipulation operators are required [16, 15]. The model we use in this paper is the one proposed by [2] and called the *Verso model* which is based on *partitioned normal form (PNF) relations* [14]. The reason for adopting this model is because of its flexibility in supporting empty sets, the assumption that relations are in partitioned normal form (which has clearer semantics than general nested relations), and its ability to allow partial updates. Also, some commercial object-relational database systems such as Informix support the use of PNF relations.

The main contributions of this paper are as follows. Firstly, we derive incremental expressions for the data manipulation operators in the Verso model. Interestingly, these expressions differ significantly from those derived for the *flat* relational model [13]. Secondly, we propose an algorithm to propagate base relation updates to a materialized view when the view is defined as a complex nested relational algebra expression. Lastly, we implement our view maintenance algorithm and perform experiments to determine what we call the *maintenance limit* of our algorithm, which is defined to be the limit on the size of the update beyond which incremental maintenance is no longer cheaper than full view recomputation. This is an important issue and one that up to now has not been adequately investigated in the literature.

The rest of this paper is organized as follows. In Section 2, we introduce the Verso model and its operators. In section 3, we define containment and disjointedness properties for the PNF nested relations. These two properties will be used in Section 4 for deriving IEs. Section 4 contains IEs derived for PNF nested operators and the derivation proofs. In Section 5, we propose a view maintaining algorithm that maintains a view using IEs when the view is defined with multiple operators. Section 6 covers the implementation details of the IEs and performance analysis. In the last section of the paper, we give the conclusion.

2 Data Model and Operators

In this section, we review the Verso data model and algebra defined in [2].

2.1 Trees

A *tree* T is a finite, acyclic, directed graph in which there is a unique node, called the *root* and denoted by $root(T)$, with in-degree (the number of edges coming into the node) 0 and every other node has in-degree 1.

A node n' is a child of a node n (or equivalently, n is the *parent* of n') if there is a directed edge from n to n'.

A node is a *leaf* if it has no children.

The *level* of a node n in a tree T is the number of nodes on a path from the root of T to n. Thus, the level of the root node is 1 and the root node is said on the *top level.*

The *height* of a tree is the maximum level of any node in the tree.

A tree T' is a *subtree* of a tree T if the nodes of T' are a subset of those of T and for every pair of nodes n' and n, n' is a child of n in T if and only if n' is a child of n in T'.

A subtree T' is a *child subtree* of T if the root node of T' is a child of T and the set of all nodes of T' and the set of all nodes of the child of T are equivalent.

2.2 Schema Trees and Nested Relation Schemas

Let U be a fixed countable finite set of atomic attribute names. Each attribute name $A \in U$ is associated with a countably infinite set of values denoted by $dom(A)$.

A *schema tree* T is a tree having at least one node; *each* node of the tree is labeled by a set of names from U. The names on the labeled nodes form a partition of U.

A *nested relation schema* is the set of attribute names mapped from a schema tree T, denoted by $sch(T)$, and defined recursively by:

(i) If T contains only one node (the root), then $sch(T) = \{A_1, ..., A_m\}$ where $A_1, .., A_m$ are attributes labeled on the root of T;

(ii) If $T_1, ..., T_n$ are child subtrees of T and $A_1, .., A_m$ are attributes labeled on the root of T, then $sch(T) = \{A_1, ..., A_m, sch(T_1), \ldots, sch(T_n)\}$.

In the schema definition, $A_1, ..., A_m$ are called the *atomic attributes* while $sch(T_1), ..., sch(T_n)$ are called the *structured attributes*. We denote each structured attribute $sch(T_i)$ $(i = 1, ..., k)$ by R_i^* and simplify $sch(T)$ by R. As a result, $sch(T) = R = \{A_1, \ ..., \ A_m, \ R_1^* : sch(T_1), \ ..., \ R_i^* : sch(T_i), \ ..., \ R_n^* : sch(T_n)\}$. Note that R_i^* is used only for referencing the schema of the child tree. If necessary, R_i^* can be labeled at the edge from $root(T)$ to $root(T_i)$.

Let $R' = sch(T')$ and $R = sch(T)$. R' is a *subschema* of R, denoted by $R' \subseteq R$, if T' is a subtree of T. The *level of an attribute* in R is defined to be the level of the node in the tree where the attribute is labeled. When the level l of an attribute

is specially concerned, l is attached to the attribute name as a superscript: A_i^l or R_j^{*l}. The *levels of the schema* R is defined to be the height of T. If a schema has l levels, the schema is called a *l-level nested* schema.

Because nested relation schemas are sets, set operations of union ($\cup$), difference ($-$), and intersection ($\cap$) can be applied to the top levels of schemas. Subset ($\subseteq$) can also be defined on the top levels of two schemas.

We define some short-hand notations for schemas. The set of atomic attributes on the top level of R is denoted by $\alpha(R)$ which is $\{A_1, ..., A_m\}$. The set of all structured attributes on the top level of R is denoted by $\beta(R)$ which is $R - \alpha(R)$. The function $\alpha_U(R)$ is defined to return all atomic attribute names labeled on all nodes of the schema tree of R.

The following is an example of a nested relation schema.

Example 2.1. We introduce a nested relation schema for a student database. A student with the name of $Name$ has studied some subjects $Subjs^*$. The student has achieved a set of marks (denoted by $Marks^*$) for each subject; each mark is for a different test type of the subject. The student also has a set of telephone numbers stored in the database for the convenience of communication. The schema tree describing the student data is given in Figure 1. The schema of the schema tree is $Stud = \{Name, Subjs^*:\{sjName, Year, Marks^*:\{testName, Mark\}\}, Tel^*:\{Tel\}\}$.

The schema is a three-level nested relation schema. On the first level, there is one atomic attribute $Name$ and two structured attributes (structured attributes) $subjs^*$ and $Tels^*$. That is, $\alpha(Stud) = \{Name\}$ and $\beta(Stud) = \{subjs^*, Tels^*\}$. The set of all atomic attributes of the schema is $\alpha_U(Stud) = \{Name, sjName, Year, Tel, testName, Mark\}$.

A subschema of $Stud$ is $sjTest = \{sjName, tests^* : \{testName\}\}$ or $studTel = \{Name, Tels^* : \{Tel\}\}$.

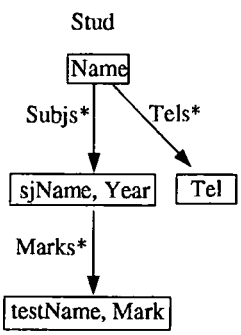

Figure 1: Schema tree *Stud*

Now we define the notion of prime subschema.

Definition 2.1 (Prime Subschema). *Let* $R = \alpha(R)\{R_1^*, ..., R_n^*\}$ *and* $S = \alpha(S)\{S_1^*, ..., S_{ns}^*\}$. *S is a* prime subschema *of R, denoted by* $S \subseteq^p R$, *if*

(1) $\alpha(R) = \alpha(S)$ *and* $\beta(R) = \beta(R) = \phi$;

(2) $\alpha(R) = \alpha(S)$ *and for each* S_k^* $(k \in [1, ..., ns])$, *there exists a* R_j^* $(j \in [1, ..., nr])$ *such that* S_k^* *is the prime subschema of* R_j^*.

Note that if S is a prime subschema of R, then $\alpha(S) = \alpha(R)$ and the definition is recursive, which means that on each level of the two schemas, two corresponding structured attributes share the same atomic attribute set. The next example shows a prime subschema.

Example 2.2. Let $StudTel = \{Name, Tels^* : \{Tel\}\}$. Then *StudTel* is a prime subschema of *Stud* defined in Example 2.1 because the two schemas have the same set of atomic attributes $\{Name\}$ on the top level and because $Tels^*$ in *StudTel* is the same as $Tels^*$ in *Stud*.

2.3 Nested Relations

We now recursively define the domain of a schema R, denoted by $dom(R)$, by:

(i) If R is of one level, $dom(R) = dom(A_1) \times \ldots \times dom(A_m)$;

(ii) If R is of more than one level, then $dom(R) = dom(A_1) \times \ldots \times dom(A_m) \times P(dom(R_1^*)) \times \ldots \times P(dom(R_n^*))$ where $P(D)$ denotes the set of all nonempty, finite subsets of a set D.

A *nested relation* over a nested relation schema $R = \{A_1, ..., A_m, R_1^*, ..., R_n^*\}$, denoted by $r(R)$, or often simply by r when R is understood, is defined to be a finite set of elements from $dom(R)$. An element t in a relation is called a *tuple* and has the form of $t =< a_1, ..., a_m, r_1, ..., r_n >$ where $a_i \in dom(A_i)$ and r_j, called a *subrelation*, is a relation over the definition of structured attribute R_j^*. Each item, a_i or r_j, is called a *value* or a *component*. Two tuples are equivalent if their corresponding components are equivalent.

The *restriction* of tuple t to attributes A_i and to R_j^*, denoted by $t[A_i]$ and $t[R_j^*]$ respectively, is defined to be $t[A_i] = a_i$ and $t[R_j^*] = r_j$. If $Y = \{A_{1y} ... A_{my} R_{1y}^* ... R_{ny}^*\}$ is a subset of R, the **restriction of t to the subset Y**, denoted by $t[Y]$, is defined to be a tuple $< t[A_{1y}], ...t[A_{my}], t[R_{1y}^*], ...t[R_{ny}^*] >$. The **restriction** of relation r to Y, denoted by $r[Y]$, is defined to be the nested relation $\{t[Y] | t \in \mathbf{r}\}$.

We now give an example of a nested relation.

Example 2.3. Let *Stud* be the nested relation schema defined in Example 2.1. A nested relation r over the schema *Stud* is given in Table 1. There are three tuples in the relation: two tuples are for student *Jack* and one for *John*. Subrelations are labeled by pairs of curly brackets.

A nested relation is in *Partitioned Normal Form*(PNF) if all atomic attributes on the top level of the relation comprise the key and all subrelations are in partitioned normal form [14]. The nested relation in Table 1 is a PNF nested relation.

Table 1: A nested relation *stud* on schema *Stud*

Name	*Subjs**				*Tels**
	sjName	*Year*	*Marks**		*Tel*
			testName	*Mark*	
Jack	*DB*	1998	*test1*	81	04143
			test2	90	1435
			exam	80	2302
	Java	1997	*ass1*	60	
			exam	80	
John	DB	1998	*test1*	60	{ 2354 }
			test2	80	

2.4 Verso Operators

In this section we review the definitions of the Verso operators proposed in [2] and reviewed in [8].

Definition 2.2 (Expansion Operator). *Let S be a prime subschema of R. Let s be a relation defined over S. The* expansion *of s to schema R is a relation over R, denoted by $\eta_R(s)$, is defined recursively by:*

$$\eta_R(s) = \{x | \exists\, v \in s \ \wedge\ x[\alpha(R)] = v[\alpha(R)] \ \wedge\ \forall i \in [1,...,n] \\ (\textit{if } \exists\, S_j^* \ \wedge\ S_j^* \subseteq R_i^* (x[R_i^*] = \eta_{R_i}(v[S_j^*])) \\ \textit{else } x[R_i^*] = \phi \qquad\qquad) \qquad\qquad \}$$

The expansion operator recursively packs each tuple in s with empty sets to make it match the schema of R. The next example shows the use of the operator.

Example 2.4. Let $s = \{< Tony, \{51234, 51535\} >\}$ be a relation on schema $S = \{Name, Tels^* : \{Tel\}\}$. Let *Stud* be the schema described in Example 2.3. Then, $\eta_{Stud}(s) = \{< Tony, \phi, \{51234, 51535\} >\}$ where the empty set ϕ is the value packed for structured attribute $Subjs^*$.

Definition 2.3 (Projection Operator). *Let S be a prime subschema of schema R. Let r be a relation defined over R. The* projection *of r onto S is a relation over S, denoted by $\mathring{\pi}_S(r)$, defined recursively by:*

(i) $\mathring{\pi}_S(r) = \{x | x \in r\}$, *if R is flat;*

(ii) $\mathring{\pi}_S(r) = \{x | \forall u \in r \ (\ x[\alpha(R)] = u[\alpha(R)] \ \wedge\ \forall i \in [1,...,ns] \\ (x[S_i^*] = \mathring{\pi}_{S_i^*}(u[R_j^*]) \textit{ where } S_i^* \subseteq^p R_j^* \ ) \ \}$

The projection operator preserves key values of r on every level and recursively projects subrelations of r. Following is an example showing the use of the projection operator.

Example 2.5. Let *pstud* be the relation defined as in Table 1. Let $StudTel = \{Name, Tel^* : \{Tel\}\}$. The projection of *pstud* to *StudTel*, i.e. $\mathring{\pi}_{StudTel}(pstud)$ is given in Table 2.

Table 2: Projection of *pstud* to *StudTel*

Name	*Tels**
	Tel
Jack	{ 04143, 1435, 2302 }
John	{ 2354 }

We now define the selection condition for the Verso selection operator.

Definition 2.4 (Atomic condition). *An* atomic condition c_a *over a schema* $R = \{A_1, ..., A_m, R_1^*, ..., R_n^*\}$ *is defined by* $c_a = A_i \theta a_i$ *where* $A_i \in \{A_1, ..., A_m\}$, $a_i \in dom(A_i)$, *and* $\theta \in \{<, \leq, >, \geq, =, \neq\}$.

An atomic condition is set to an atomic attribute on the top level of a schema.

Definition 2.5 (Basic condition). *A* basic condition c_b *over a schema* $R = \{A_1, ..., A_m, R_1^*, ..., R_n^*\}$ *is defined by connecting a set of atomic conditions with* $\vee$ *(or),* $\wedge$ *(and),* $\neg$ *(not), and brackets.*

Definition 2.6 (Selection condition). *A* selection condition c *over a schema* $R = \{A_1, ..., A_m, R_1^*, ..., R_n^*\}$ *is defined recursively by*

(i) $c = (c_b)$ *if* R *is flat;*

(ii) $c = (c_b \wedge c_{r1} : R_1^*.c_1 \theta' r_1 \wedge ... \wedge c_{rn} : R_n^*.c_n \theta' r_n)$.
In the condition c, c_{rj} $(j = 1, ..., n)$ *is a reference name to the expression* $R_j^*.c_j \theta' r_j$ *and ':' means 'defined by'. In* $R_j^*.c_j \theta' r_j$, $R_j^*.c_j$ *denotes the returned set selected from the subrelation over* R_j^* *by recursively applying selection condition* c_j. *The returned set then participates in the evaluation of* $\theta' r_j$ *where* r_j *is either the empty set* ϕ *or the any set* ω [1] *over* R_j^*. *When* r_j *is* ϕ, θ' *is one of* $\{=, \neq\}$ *while when* r_j *is* ω, θ' *is* $=$.
We call c_{rj} *the* existence condition *on subrelation of* R_j^*.

We now give an example of a selection condition.

Example 2.6. For schema *Stud* = {*Name*, *Subjs**:{*sjName*, *Year*, *Marks**: {*testName*, *Mark*}}, *Tel**: {*Tel*}} defined in Example 2.1, a select condition over the schema is $c = (Name =' Jack' \wedge Subjs^* : (Marks^* : (Mark \geq 90) \neq \phi) \neq \phi)$. This selection condition selects a student named '*Jack*' who has obtained at least a good mark (≥ 90) for some subjects.

We use $c_b(x[\alpha(R)]) = true$ to denote the case where the key value of a tuple x makes an existence condition true. Accordingly, we use $c_{rj}(x[R_j^*]) = true$ to mean the case where a subrelation on R_j^* makes an existence condition is true.

[1] 'Any set' means that the number of elements in the set does not matter.

Definition 2.7 (Selection Operator). *Let $R = \alpha(R)\{R_1^*, ..., R_n^*\}$ be a schema and r be a relation over R. Let c be a selection condition defined with Definition 2.6. The* selection *of r based on c is a relation over R, denoted by $\hat{\sigma}_c(r)$, recursively defined by:*

$$\begin{aligned}
&(i)\ \hat{\sigma}_c(r) = \{x | x \in r \text{ and } c_b(x[\alpha(R)]) = true\}, \text{ if } R \text{ is flat;}\\
&(ii)\ \hat{\sigma}_c(r) = \{x | \exists\, u \in r \ \wedge\ c_b(x[\alpha(R)]) = true \ \wedge\ x[\alpha(R)] = u[\alpha(R)] \ \wedge\\
&\qquad\qquad \forall\, j \in (1, ..., n)(c_{rj}(x[R_j^*] = \hat{\sigma}_{c_j}(u[R_j^*])) = true\ )\qquad \}
\end{aligned}$$

Example 2.7. We apply the selection to relation *stud* in Table 1 with the selection condition c defined in Example 2.6. The returned relation from the selection is given in Table 3. We take the second tuple, denoted by t_2 in *stud* as an example to explain the operation. $t_2[Name]$ is 'Jack', which makes the basic condition $Name =' Jack'$ true. In the recursive part and on the inner-most level, the evaluation of $(Mark \geq 90)$ against $Marks^*$ is ϕ since the mark in tuple of $Marks^*$ is less than 90. Therefore $Marks^*.(Mark \geq 90) \neq \phi$ is evaluated 'False'. Since $t_2[Subj^*]$ has only one tuple and its subrelation is evaluated to 'False', so no tuple in $t_2[Subj^*]$ can be selected. This makes $Subjs^*.(Marks^*.(Mark \geq 90) \neq \phi) \neq \phi$ 'False'. As a result, the evaluation of the selection condition against this tuple is 'False' and not in Table 3.

Table 3: $\bar{\sigma}_c$ (*stud*)

Name	*Subjs**				*Tels**
	sjName	*Year*	*Marks**		*Tel*
			testName	*Mark*	
Jack	*DB*	1998	*test2*	90	04143 1435

Definition 2.8 (Union Operator). *Let r and s be two relations over R. The* union *of r and s is a relation over R, denoted by $r \oplus s$, and recursively defined by:*

$$\begin{aligned}
&(i)\ r \oplus s = \{x | x \in r \text{ or } x \in s\}, \text{ if } R \text{ is flat;}\\
&(ii)\ r \oplus s = \{x | \exists\, u \in r \ \wedge\ \exists\, v \in s \ \wedge\ x[\alpha(R)] = u[\alpha(R)] = v[\alpha(R)] \ \wedge\\
&\qquad\qquad \forall\, i \in [1, ..., n](\ x[R_i^*] = u[R_i^*] \oplus v[R_i^*]\ ) \text{ or}\\
&\qquad \exists\, u \in r \ \wedge\ x[\alpha(R)] = u[\alpha(R)] \notin s[\alpha(R)] \ \wedge\ x = u\ ) \text{ or}\\
&\qquad \exists\, v \in s \ \wedge\ x[\alpha(R)] = v[\alpha(R)] \notin r[\alpha(R)] \ \wedge\ x = v\ )\ \}
\end{aligned}$$

The union operator recursively combines two tuples, one from each operand relation, if their key values match on each level. The operation guarantees that the output of the union is in PNF, i.e., there are no duplicate values for atomic attributes on each level of the relation. The next example introduces the use of the union operator.

Example 2.8. Table 5 shows the union of relation *pstud* in Table 1 and relation $\delta pstud$ in Table 4. *pstud* and $\delta pstud$ each has a tuple with key value of *Jack*. As a result, the subrelations of the two *Jack* tuples are combined. This rule is applied recursively until two *ass*1 tuples on the most internal level of *java* 1997 merges into one tuple in the union and *ass*2 of *java* 1997 is added to the union. The same combination applies to $Tels^*$.

Tuple *John* in *pstud* and tuple *Andrew* in $\delta pstud$ do not match any tuples in the other relation, they appear the same as they were before the union.

Table 4: $\delta pstud$

Name	$Subjs^*$				$Tels^*$
	sjName	*Year*	$Marks^*$		*Tel*
			testName	*Mark*	
Jack	*Java*	1997	*ass*1	60	{ 54111 }
			*ass*2	70	
Andrew	{ *DB*	1999	{ *test*1	60 } }	ϕ

Table 5: Union of *pstud* and $\delta pstud$

Name	$Subjs^*$				$Tels^*$
	sjName	*Year*	$Marks^*$		*Tel*
			testName	*Mark*	
Jack	*DB*	1998	*test*1	81	{ 54111
			*test*2	90	04143
			exam	80	1435
	Java	1997	*ass*1	60	2302 }
			*ass*2	70	
			exam	80	
John	DB	1999	*test*1	60	{ 2354 }
			*test*2	80	
Andrew	{ *DB*	1999	{ *test*1	60 } }	ϕ

Definition 2.9 (Difference Operator). *Let r and s be two relations over R. The* difference *of r and s is a relation over R, denoted by $r \ominus s$, and recursively defined by:*

$$\begin{aligned}
&(i)\ \ r \ominus s = \{x | x \in r \text{ and } x \notin s\}, \text{ if } R \text{ is flat;}\\
&(ii)\ \ r \ominus s = \{x | \exists\, u \in r \wedge \exists\, v \in s \ \wedge\ x[\alpha(R)] = u[\alpha(R)] = v[\alpha(R)] \wedge\\
&\qquad\qquad \forall\, i \in [1, ..., nt](\ x[R_i^*] = u[R_i^*] \ominus v[R_i^*] \neq \phi\ ) \text{ or}\\
&\qquad \exists\, u \in r \ \wedge\ x[\alpha(R)] = u[\alpha(R)] \notin s[\alpha(R)] \ \wedge\ x = u\ \}
\end{aligned}$$

The difference operator is like the union operator in that it recursively differences subrelations if the key values of two tuples, one from each operand relation,

match. The output of the difference operator is a relation in PNF. The following example shows the use of the difference operator.

Example 2.9. Table 6 gives the difference of *pstud* in Table 1 and the relation $\delta pstud$ in Table 4. The difference is applied to the two tuples having the name of *Jack* in the two relations. This procedure recursively applies until it reaches the most inner level. As a result, in the most inner level the tuple *ass*1 of *Java* 1997 does not appear in the result. The tuple *John* in relation *pstud* does not match any tuples in the relation $\delta pstud$ and it appears the same in output. In contrast, the tuple *Andrew* in relation $\delta pstud$ does not affect any tuple in *pstud* because of no match of key values and is excluded in the result.

Table 6: $pstud \ominus \delta pstud$

Name	*Subjs**				*Tels**
	sjName	*Year*	*Marks**		*Tel*
			testName	*Mark*	
Jack	*DB*	1998	*test*1	81	04143
			*test*2	90	1435
			exam	80	2302
	Java	1997	*exam*	80	
John	DB	1999	*test*1	60	{ 2354 }
			*test*2	80	

Definition 2.10 (Intersection Operator). *Let r and s be two relations over R. The* intersection *of r and s is a relation over R, denoted by $r \odot s$, and recursively defined by:*

(i) $r \odot s = \{x | x \in r \text{ and } x \in s\}$, *if R is flat;*

(ii) $r \odot s = \{x | \exists u \in r \wedge \exists v \in s \ (\ x[\alpha(R)] = u[\alpha(R)] = v[\alpha(R)] \wedge \forall i \in (1, ..., n) \ (\ x[R_i^*] = u[R_i^*] \odot v[R_i^*]\ )\ \}$

The use of the intersection operator is shown in the next example.

Example 2.10. Table 8 shows the intersection of $pstud_1$ in Table 7 and *pstud* in Table 1.

Table 7: A nested relation $pstud_1$

Name	*Subjs**				*Tels**
	sjName	*Year*	*Marks**		*Tel*
			testName	*Mark*	
Jack	{ *DB*	1998	{ *exam*	80 } }	83304143

Table 8: *pstud* $\odot$ *pstud*$_1$

Name	*Subjs**				*Tels**
	sjName	*Year*	*Marks**		*Tel*
			testName	*Mark*	
Jack	{ *DB*	1998	{ *exam*	80 } }	ϕ

Definition 2.11 (Joinable schemas). *Let S and R be two schemas satisfying $\alpha(R) = \alpha(S)$. Then R and S are* joinable schemas *if there exists a schema T such that (1) $\alpha_U(T) = \alpha_U(R) \cup \alpha_U(S)$; (2) both R and S are prime subschemas of T. We call T the* ***joined schema****.*

Example 2.11. Let *Stud* be a schema defined in Example 2.3. Let $S = \{Name, Addrs^* : \{Addr\}\}$ be another schema which describes the student addresses. *Stud* and S are joinable because $\alpha(Stud) = \alpha(S)$ and there exists a schema $T = \{Name, Subjs^* : \{Subj, Year, Marks^* : \{TestName, Mark\}\}, Tels^* : \{Tel\}, Addrs^* : \{Addr\}\}$ such that (1) $\alpha_U(T) = \alpha_U(Stud) \cup \alpha_U(S)$; (2) *Stud* and S are prime subschemas of T. So T is the joined schema.

Definition 2.12 (Join). *Let S and R be joinable schemas and T be the joined schema. Let r and s be relations on R and S respectively. The* join *of r and s is a relation over T, denoted by $r \mathring{\bowtie} s$, defined recursively by:*

(i) $r \mathring{\bowtie} s = \{x | x \in r \ \wedge \ x \in s\}$, *if $R = S = T$ are flat;*

(ii) $r \mathring{\bowtie} s = \{x | \exists\, u \in r \ \wedge \ \exists\, v \in s \ \wedge \ x[\alpha(R)] = u[\alpha(R)] = v[\alpha(R)] \ \wedge \ \forall\, i \in [1, ..., nt]$
$\quad$ *(if* $\exists\, R_j^* \underline{\subseteq}^p T_i^* \ \wedge \ \exists\, S_k^* \underline{\subseteq}^p T_i^* \ (\, x[T_i^*] = u[R_j^*] \mathring{\bowtie} v[S_k^*] \,)$ *or*
$\quad$ *if* $\exists\, R_j^* \underline{\subseteq}^p T_i^* \ \wedge \ \nexists\, S_k^* \underline{\subseteq}^p T_i^* \ (\, x[T_i^*] = u[R_j^*])$ *or*
$\quad$ *if* $\exists\, S_k^* \underline{\subseteq}^p T_i^* \ \wedge \ \nexists\, R_j^* \underline{\subseteq}^p T_i^* \ (\, x[T_i^*] = v[S_k^*])$ $\quad\quad)\}$

The join operator joins two relations based on the equivalence of the values of the atomic attributes starting from the top level. The next example shows the use of the join operator.

Example 2.12. Let *pstud* be defined in Table 1. Let *studAddr* be a relation defined in Table 9. The results of join of *pstud* and *studAddr* is shown in Table 10.

Table 9: *studAddr*

Name	*Addrs**
	Addr
John	{12 *Newton st*, 5 *Darling av* }

The Verso operators presented in this section have the property of preserving key attributes on all levels. In other words, all operators do not shrink or expand keys of relations. For example, the projection operation only projects structured attributes but not atomic attributes. This property guarantees the results of operations are in partitioned normal form.

Table 10: The join of *pstud* and *stAddr*

Name	*Subjs**				*Tels**	*Addrs**
	sjName	*Year*	*Marks**		*Tel*	*Addr*
			testName	*Mark*		
John	DB	1999	*test1*	60	{ 2354 }	12 *Newton st*
			test2	80		5 *Darling av*

3 Containment and Disjointedness in Nested Relations

In this section, we review the definitions and results from [11] concerning the properties of containment and disjointedness in PNF relations. These results will be used in deriving IEs in the next section. At the same time, we compare the properties of containment and disjointedness for nested relations with the corresponding properties in flat relations.

In flat relations [13, 4], disjointedness means that when an insertion is made to a relation, the tuples to be inserted should not be included in the relation; whereas containment means that when tuples are deleted from a relation, the deleted tuples should be contained in the relation. It is also desirable in many applications, such as those involving triggers or real-time databases, that the changes to the view computed using IEs also satisfy the containment and disjointedness properties.

The issue of how to extend the definitions of containment and disjointedness from flat relations to PNF relations is not as straightforward as might first appear. This is discussed in more detail in [11] but we briefly summarise our approach here for the sake of completeness. In [11] we adopted the approach of [10, 15]. In this approach we require that the definitions for containment for and disjointedness must be *faithful* and *precise*. By faithful, we mean that the definitions for containment and disjointedness for PNF relations should coincide with the definitions for containment and disjointedness for flat relations when the PNF relations are in fact flat. By preciseness we mean that the properties should coincide with the corresponding properties for flat relations when applied to the total unnnests of the PNF relations.

For containment, we proposed the following definition in [11] and showed it to be faithful and precise.

Definition 3.1 (Containment). *Let r and δr be two instances over schema R. Then δr is defined to be contained in r, denoted by $\delta r \underline{\subseteq} r$, if:*

(i) when R is flat, $\forall v \in \delta r \ \wedge \ v \in r$;

(ii) when R is not flat, $\forall v \in \delta r \ \wedge \ \exists u \in r \ \wedge \ v[\alpha(R)] = u[\alpha(R)] \ \wedge \ \forall i \in [1, ..., nr](v[R_i^*] \underline{\subseteq} u[R_i^*])$.

For example in Table 11, $\delta r \underline{\subseteq} r$. However, we note that in this table that δr is not a subset of r.

Table 11: Relations showing the containment

A	B^*
	B
a_1	$\{b_1\}$

δr

A	B^*
	B
a_1	$\{b_1, b_2\}$

r

Also, in [11] we show that nested containment has the following properties. These properties will be used in the next section.

Theorem 3.1. *Let r and δr be two instances over schema R. Then the following are equivalent:*

(i) $\delta r \,\underline{\subseteq}\, r$;

(ii) $r \odot \delta r = \delta r$;

(iii) $r \oplus \delta r = r$.

As for disjointedness, the following definition was proposed in [11] and shown to be faithful and precise.

Definition 3.2 (Disjointedness). *Let $\delta r \neq \phi$ and r be two relations over schema R. δr is defined to be disjoint from r, denoted by $\delta r \,|\overline{\subseteq}\, r$, if*

(i) r *is* ϕ;

(ii) when R is flat, $\forall v \in \delta r \ \wedge \ v \notin r$;

(iii) when R is not flat, $\forall v \in \delta r$,

(a) $v[\alpha(R)] \notin r[\alpha(R)]$ *or*

(b) $\exists u \in r$, $v[\alpha(R)] = u[\alpha(R)]$ *and*
$\exists i (v[R_i^*] \neq \phi \ \wedge \ v[R_i^*] \,|\overline{\subseteq}\, u[R_i^*])$.

For example, the two relations shown in Table 12 are disjoint.

Table 12: Two cases of disjointedness

A	B^*	C^*
	B	C
a	$\{b_1\}$	$\{c_1\}$

δr

A	B^*	C^*
	B	C
a	$\{b_2\}$	$\{c_2\}$

r

We now introduce another type of disjointedness which, when it holds, we will show in the next section to considerably simplify incremental equations.

Definition 3.3. *Let r_1 and r_2 be two nested relations defined over schema R and let $A \subseteq R$. Then a tuple $x \in r_1$ is* A-disjoint *from r_2 if $x[A]$ is not in $r_2[A]$ (otherwise x is said* A-overlapping *with r_2). The two relations r_1 and r_2 are defined to be* A-disjoint *if every $x \in r_1$ is A-disjoint from r_2 (note that the definition is symmetric).*

We now illustrate the definition by Example 3.1.

Example 3.1. There are three relations r_0, r_1, and r_2 defined over a schema $R = \{A, B, C^* : \{C\}, D^* : \{D\}\}$ in Table 13. Let $Y = \{B, C^*\}$. Then r_0 and r_1 are $R - Y$ disjoint, but r_0 and r_2 are not. This is because $R - Y = \{A, D^*\}$ while $r_0[\{A, D^*\}] \cap r_1[\{A, D^*\}] = \phi$ and $r_0[\{A, D^*\}] \cap r_2[\{A, D^*\}] = \{< a_1, \{d_1, d_2\} >\} \neq \phi$.

The first tuple in r_0 is a $R - Y$ overlapping tuple with r_2 while the second tuple in r_0 is a $R - Y$ disjoint tuple with r_2.

Table 13: An example for $R - Y$ disjointedness

A	B	C^*	D^*
		C	D
a_1	b_1	$\{c_1, c_2\}$	$\{d_1, d_2\}$
a_2	b_1	$\{c_1, c_2\}$	$\{d_1, d_2\}$

r_0

A	B	C^*	D^*
		C	D
a_1	b_2	$\{c_1, c_2\}$	$\{d_1\}$

r_1

A	B	C^*	D^*
		C	D
a_1	b_2	$\{c_1, c_2\}$	$\{d_1, d_2\}$

r_2

4 Incremental Equations for Nested Operators

In this section, we derive incremental expressions for the nested operators defined in Section 2. We assume that the update to a relation is a full tuple update, i.e., the updating tuples and the relation have the same schema. Otherwise, if the schema of the update is a prime subschema of the updated relation, we assume that the expansion operator has been applied to expand the updating tuples into full tuples.

We firstly give a general overview of what we are aiming to derive in this section of the paper. We are aiming to derive equations of the form $op_u(r \oplus\!\!\!\!\bigcirc \delta r) = f(op_u(r), r, \delta r)$ in the case of a unary query operator op_u, and $op_b(r \oplus\!\!\!\!\bigcirc \delta r, s) = f(op_b(r, s), r, s, \delta r)$ in the case of a binary operator op_b. In this notation $\oplus\!\!\!\!\bigcirc$ means either the PNF union operator $\oplus$ or the PNF difference operator $\ominus$; r and s are called ***base relations***; δr is called the ***update*** to the base relation and f is a function. We call $op_u(r)$ and $op_b(r, s)$ the ***old views***, $op_u(r \oplus\!\!\!\!\bigcirc \delta r)$ and $op_b(r \oplus\!\!\!\!\bigcirc \delta r, s)$ the ***recomputation***, $f(op_u(r), r, \delta r)$ and $f(op_b(r \oplus\!\!\!\!\bigcirc s), r, s, \delta r)$ the ***incremental computation***. For each equation, we use the abbreviation of LHS for left hand side and RHS for right hand side.

It is particularly desirable if the RHS of the IE for an operator take the simple form of $op_u(r) \oplus\!\!\!\!\bigcirc op_u(\delta r)$ $(op_b(r, s) \oplus\!\!\!\!\bigcirc op_b(\delta r, s))$. We call this form of IE the ***standard form***. The advantage of this form is that is does not involve extra operators. When the size of the increment is small, in general it is much more efficient to compute the new view incrementally than by recomputation. Standard

IEs may not exist for some operators, but we can in some cases still derive IEs in the ***limited standard form*** which means a standard form attached with some conditions. The advantage of the limited standard form of an IE is that it reveals the reason why the IE can not be standard. However, the test of conditions in the limited standard form can be costly because recursive traversal down to subrelations is needed and there is no index possible for the internal subrelations. To avoid testing the expensive conditions, we define the ***implementation form*** for IEs. In the implementation form, the concept of the attribute disjointedness defined in the last section is used and testing the expensive conditions is replaced by top level selection.

We note that we use induction to prove the IEs in the section because all the operators involved in IEs of the section are recursive. In the proofs of induction, we will firstly prove that an equation is correct for a flat relation and then prove the equation is correct for a n-level nested relation if it holds for $(n-1)$-level nested subrelations.

4.1 Incremental Equations for the Expansion Operator

Theorem 4.1. *Let S be a prime subschema of a schema R and let r and δr be two instances over S. Then the following two expressions for the expansion operator are true.*

$$\eta_R(r \oplus \delta r) = \eta_R(r) \oplus \eta_R(\delta r) \tag{1}$$

$$\eta_R(r \ominus \delta r) = \eta_R(r) \ominus \eta_R(\delta r) \tag{2}$$

Proof.

Proof of Equation 1:

(1) Base Case: when $R = S$ are flat, the equation holds. The proof is obvious. In this case, by the definition of expansion, on LHS: $\eta_R(r) = r$, $\eta_R(\delta r) = \delta r$, $\eta_R(r) \oplus \eta_R(\delta r) = r \oplus \delta r$. on RHS: $\eta_R(r \oplus \delta r) = r \oplus \delta r$. Base case is proved.

(2) Induction: suppose $\eta_{R_i^*}(u[S_i^*] \oplus v[S_i^*]) = \eta_{R_i^*}(u[S_i^*]) \oplus \eta_{R_i^*}(v[S_i^*])$ where $u \in r$ and $v \in \delta r$. We prove the equation is correct over r and δr.

(a) $\eta_R(r \oplus \delta r) \subseteq \eta_R(r) \oplus \eta_R(\delta r)$
For a tuple $x \in \eta_R(r \oplus \delta r)$, by the definition of union, x is expanded from a tuple u of r, a tuple v of δr, or a tuple unioned from u and v.

(i) x is expanded from u (i.e. $u[\alpha(R)] \notin \delta r[\alpha(R)]$):
$x = u[\alpha(R)](\eta_{R_1^*} u[S_1^*])...(\eta_{R_m^*} u[S_m^*]) \underbrace{\phi \ ... \ \phi}_{m+1...nr}$.
On RHS : since $u \in r$, the expansion of u, which is the same as x, is contained in $\eta_R(r)$. Because expansion does not change key values of tuples, $u[\alpha(R)] \notin \delta r[\alpha(R)] \Longrightarrow u[\alpha(R)] \notin \eta_R(s)[\alpha(R)]$. Further, the union in RHS does not change values of the tuple expanded from u. Hence, $x \in (\eta_R(r) \oplus \eta_R(\delta r))$.

(ii) x is expanded from v: this case is symmetric to the last case.

(iii) x is expanded from the union of u and v ($u[\alpha(R)] = v[\alpha(R)]$): The union of u and v is $u[\alpha(R)](u[S_1^*] \oplus v[S_1^*])...(u[S_m^*] \oplus v[S_m^*])$. Then x is expanded from this union, $x = u[\alpha(R)](\eta_{R_1^*}(u[S_1^*] \oplus v[S_1^*]))...(\eta_{R_m^*}(u[S_m^*] \oplus v[S_m^*]))\ \underbrace{\phi\ ...\ \phi}_{m+1...nr}$. By the induction assumption that the equation holds on level $(n-1)$, we have union and expansion are exchangeable on second level. Then $x = u[\alpha(R)](\eta_{R_1^*}u[S_1^*] \oplus \eta_{R_1^*}v[S_1^*])...(\eta_{R_m^*}u[S_m^*] \oplus \eta_{R_m^*}v[S_m^*])\ \underbrace{\phi\ ...\ \phi}_{m+1...nr}$. By rewriting, $x = u[\alpha(R)](\eta_{R_1^*}u[S_1^*] \oplus \eta_{R_1^*}v[S_1^*])...(\eta_{R_m^*}u[S_m^*] \oplus \eta_{R_m}v[S_m^*])\ \underbrace{(\phi \oplus \phi)...(\phi \oplus \phi)}_{from\ m+1\ to\ nr}$ By definition of union and expansion, x is the union of the expansion of u and the expansion of v. So the tuple x is contained in RHS.

Item (a) is proved.

(b) $\eta_R(r) \oplus \eta_R(\delta r) \subseteq \eta_R(r \oplus \delta r)$ The proof is similar to Item (a) and omitted.

The equation is proved.

Intuitively, the expansion operator packs the structured attributes in R but not in S with ϕ. It changes neither values for the structured attributes nor the key values of r and δr. As a result, expansion does not affect the union property of r and δr, and the equation is correct. In other words, the expansion operator is faithful [10] with respect to the union operator.

Proof of Equation 2:

(1) Base case: when $R = S$ are flat, the equation holds. In this case, bythe definition of expansion, on LHS, $\eta_R(r) = r$, $\eta_R(\delta r) = \delta r$, $\eta_R(r) \ominus \eta_R(\delta r) = r \ominus \delta r$. On RHS, $\eta_R(r \ominus \delta r) = r \ominus \delta r$. Base case is proved.

(2) Induction: suppose $\eta_{R_i^*}(u[S_i^*] \ominus v[S_i^*]) = \eta_{R_i^*}(u[S_i^*]) \ominus \eta_{R_i^*}(v[S_i^*])$ where $u \in r$ and $v \in \delta r$. We prove the equation is correct over r and δr.

(a) $\eta_R(u[S] \ominus v[S]) \subseteq \eta_R(u[S]) \ominus \eta_R(v[S])$
For a tuple $x \in (\eta_R(u[S] \ominus v[S]))$, there must exit a tuple $x' \in (r \ominus \delta r)$ such that x is expanded from x'. By the definition of difference, x' must be produced from r and δr in two disjoint cases.

(i) x' is from r: $\exists\, u \in r$, $u[\alpha(R)] \notin \delta r[\alpha(R)]$, $x' = u$.
In this case, x is expanded from u as $x = u[\alpha(R)]\eta_{R_1}(u[S_1^*])...\eta_{R_m}(u[S_m^*])\ \underbrace{\phi\ ...\ \phi}_{m+1...nr}$. On RHS : $u \in r \Longrightarrow$ the expansion of u, which is the same as x, is contained in $\eta_{R_i^*}(u[S_i^*])$. Because expansion does not change key value of tuples, $u[\alpha(R)] \notin \delta r[\alpha(R)] \Longrightarrow u[\alpha(R)] \notin \eta_R(\delta r)[\alpha(R)]$. By the definition

of difference, the expansion of u is not changed by difference. So $x \in RHS$.

(ii) x' is the difference of tuples from r and δr: $\exists u \in r$, $\exists v \in \delta r$, $u[\alpha(R)] = v[\alpha(R)]$
$x' = u[\alpha(R)](u[S_1^*] \ominus v[S_1^*])...(u[S_m^*] \ominus v[S_m^*])$ and $\exists i \in [1,...,m], (u[S_i^*] \ominus v[S_i^*]) \neq \phi$
x is expanded from x':
$x = u[\alpha(R)](\eta_{R_1}(u[S_1^*] \ominus v[S_1^*]))...(\eta_{R_m}(u[S_m^*] \ominus v[S_m^*]))\underbrace{\phi\ ...\ \phi}_{m+1...n}$, and
$\exists i \in [1,...,m]$, $(u[S_i^*] \ominus v[S_i^*]) \neq \phi$.
Note that expansion adds empty sets to the structured attributes. It does not change the values of existing attributes. Therefore, $(u[S_i^*] \ominus v[S_i^*]) \neq \phi \Longrightarrow (\eta_{R_1} u[S_i^*] \ominus \eta_{R_1} v[S_i^*]) \neq \phi$. By rewriting x, we have
$x = u[\alpha(R)](\eta_{R_1} u[S_1^*] \ominus \eta_{R_1} v[S_1^*])...(\eta_{R_m} u[S_m^*] \ominus \eta_{R_m} v[S_m^*])\underbrace{(\phi \ominus \phi)...(\phi \ominus \phi)}_{from\ m+1\ to\ n}$ and $\exists i \in [1,...,m]$, $(\eta_{R_1} u[S_i^*] \ominus \eta_{R_1} v[S_i^*]) \neq \phi$.
This just equals to the difference of the expansion of u and the expansion of v in RHS . So we proved that $x \in RHS$.

Item (a) is proved.

(b) $\eta_{R_i}(u[S_i]) \ominus \eta_{R_i}(v[S_i]) \subseteq \eta_{R_i}(u[S_i] \ominus v[S_i])$
This item is proved in a similar way as in Item (a).

Equation 2 is proved. □

4.2 Incremental Equations for the Projection Operator

Lemma 4.1. *Let S be a prime subschema of R. Let r and δr be instances over R. Then, the following two equations hold.*

$$\mathring{\pi}_S(r \oplus \delta r) = \mathring{\pi}_S(r) \oplus \mathring{\pi}_S(\delta r) \tag{3}$$

$$\mathring{\pi}_S(r \ominus \delta r) = \mathring{\pi}_S(r) \ominus \mathring{\pi}_S(\delta r) \oplus \mathring{\pi}_S(r \ominus \delta r) \tag{4}$$

Proof.
Proof of Equation 3:

(1) Base case: when $R = S$ are flat, the projection does nothing to tuples in r and s. So the equation holds.

(2) Induction: suppose that for $u \in r$ and $v \in s$, $\mathring{\pi}_{S_j}(u[R_j^*] \oplus v[R_j^*]) = (\mathring{\pi}_{S_j}(u[R_j^*]) \oplus \mathring{\pi}_{S_j}(v[R_j^*]))$, we prove $\mathring{\pi}_S(r \oplus s) = (\mathring{\pi}_S(r) \oplus \mathring{\pi}_S(s))$.

(a) $\mathring{\pi}_S(r \oplus s) \subseteq (\mathring{\pi}_S(r) \oplus \mathring{\pi}_S(s))$
For a tuple $x \in \mathring{\pi}_S(r \oplus s)$, there must exist a tuple $x' \in (r \oplus \delta r)$ such that x is the projection of x'. x' is generated by the union in 3 cases:

(i) $\exists u \in r \wedge u[\alpha(R)] \notin \delta r[\alpha(R)]$ and x' is produced by u:
In this case, x is the expansion of u. On RHS, $u \in r \Longrightarrow x \in\in \mathring{\pi}_S(r)$; the projection does not change the key value of a tuple. Therefore no tuple in $\mathring{\pi}_S(\delta r)$ will affect $x \in \mathring{\pi}_S(r)$ when the union is conducted. $x \in (\mathring{\pi}_S(r) \oplus \mathring{\pi}_S(s))$.

(ii) $\exists v \in \delta r \wedge v[\alpha(R)] \notin r[\alpha(R)]$ and x' is produced by v: Symmetric to case (i).

(iii) $\exists u \in r \wedge \exists u \in \delta r \wedge u[\alpha(R)] = v[\alpha(R)]$ and x' is the union u and v: $x' = u[\alpha(R)](u[R_1^*] \oplus v[R_1^*])...(u[R_{nr}^*] \oplus v[R_{nr}^*])$,
x is the projection of x':
$x = u[\alpha(R)](\mathring{\pi}_{S_1^*}(u[R_1^*] \ominus v[R_1^*])...(\mathring{\pi}_{S_{ns}}(u[R_{ns}^*] \ominus v[R_{ns}^*])$
By the induction assumption,
$x = u[\alpha(R)](\mathring{\pi}_{S_1^*}(u[R_1^*]) \ominus \mathring{\pi}_{S_1^*}(v[R_1^*]))...(\mathring{\pi}_{S_{ns}}(u[R_{ns}^*]) \ominus \mathring{\pi}_{S_{ns}}(v[R_{ns}^*]))$

On RHS: let the projection of u be denoted by x^u and the projection of v be denoted by x^v. Then $x^u \in \mathring{\pi}_S(r)$ and $x^v \in \mathring{\pi}_S(\delta r)$:
$x^u = u[\alpha(R)](\mathring{\pi}_{S_1^*}(u[R_1^*]))...(\mathring{\pi}_{S_{ns}^*}(u[(R_{ns}^*)]))$
$x^v = v[\alpha(R)](\mathring{\pi}_{S_1^*}(v[R_1^*]))...(\mathring{\pi}_{S_{ns}^*}(v[(R_{ns}^*)]))$
The union of x^u and x^v produces: $u[\alpha(R)](\mathring{\pi}_{S_1^*}(u[R_1^*]) \ominus \mathring{\pi}_{S_1^*}(v[R_1^*]))...(\mathring{\pi}_{S_{ns}}(u[R_{ns}^*]) \ominus \mathring{\pi}_{S_{ns}}(v[R_{ns}^*])) \Longrightarrow x$
Consequently, $x \in (\mathring{\pi}_S(r) \oplus \mathring{\pi}_S(s))$.

Item (a) is proved.

(b) $(\mathring{\pi}_S(r) \oplus \mathring{\pi}_S(s)) \subseteq \mathring{\pi}_S(r \oplus s)$: This proof is similar to Item (a) is omitted.

The equation is proved.

Proof of Equation 4: We only need to prove that $\mathring{\pi}_S(r) \ominus \mathring{\pi}_S(\delta r) \ \underline{\subseteq}\ \mathring{\pi}_S(r \ominus \delta r)$ because of Theorem 3.1.

(1) Base case: when $R = S$ are flat, the projection does nothing to tuples in r and s. $\mathring{\pi}_S(r) \ominus \mathring{\pi}_S(\delta r) = \mathring{\pi}_S(r \ominus \delta r)$.

(2) Induction: suppose that for $u \in r$ and $v \in s$, $(\mathring{\pi}_{S_j}(u[R_j^*]) \ominus \mathring{\pi}_{S_j}(v[R_j^*])) \ \underline{\subseteq}\ \mathring{\pi}_{S_j}(u[R_j^*] \ominus v[R_j^*])$, we prove $(\mathring{\pi}_S(r) \ominus \mathring{\pi}_S(s)) \ \underline{\subseteq}\ \mathring{\pi}_S(r \ominus s)$.
For a tuple $x \in (\mathring{\pi}_S(r) \ominus \mathring{\pi}_S(s))$, x is produced in two cases:

(i) x is the difference of x^u and x_v where $x^u \in \mathring{\pi}_S(r)$ and $x^v \in \mathring{\pi}_S(\delta r)$:
Suppose x^u is the projection of $u \in r$ and x^v is the projection of $v \in r$.
By the definition of projection, we have
$x^u = u[\alpha(R)](\mathring{\pi}_{S_1^*}(u[R_1^*]))...(\mathring{\pi}_{S_{ns}^*}(u[(R_{ns}^*)]))$
$x^v = v[\alpha(R)](\mathring{\pi}_{S_1^*}(v[R_1^*]))...(\mathring{\pi}_{S_{ns}^*}(v[(R_{ns}^*)]))$
By the definition of difference,
$x = u[\alpha(R)](\mathring{\pi}_{S_1^*}(u[R_1^*]) \ominus \mathring{\pi}_{S_1^*}(v[R_1^*]))...(\mathring{\pi}_{S_{ns}}(u[R_{ns}^*]) \ominus \mathring{\pi}_{S_{ns}}(v[R_{ns}^*]))$
and $\exists i \in [1, ..., ns](\mathring{\pi}_{S_i}(u[R_i^*]) \ominus \mathring{\pi}_{S_i}(v[R_i^*]) \neq \phi)$.
From $(\mathring{\pi}_{S_i}(u[R_i^*]) \ominus \mathring{\pi}_{S_i}(v[R_i^*]) \neq \phi)$ we have $u[R_i^*] \ominus v[R_i^*] \neq \phi$ because projection makes a tuple have less attributes.

On RHS: the difference of u and v produce a tuple $y \in (r \ominus \delta r)$:
$y = u[\alpha(R)](u[R_1^*] \ominus v[R_1^*])...(u[R_{nr}^*] \ominus v[R_{nr}^*])$ since $\exists i \in [1,...,ns](u[R_i^*] \ominus v[R_i^*] \neq \phi)$.
The projection of y produces y' in RHS:
$y' = u[\alpha(R)](\mathring{\pi}_{S_1^*}(u[R_1^*] \ominus v[R_1^*])...(\mathring{\pi}_{S_{ns}}(u[R_{ns}^*] \ominus v[R_{ns}^*])$ and $\exists i \in [1,...,ns](u[R_i^*] \ominus v[R_i^*] \neq \phi)$.
By induction assumption, all subrelations of x are contained in the according subrelations of y'. Therefore, x is tuple-contained in y'.

(ii) x is in $\mathring{\pi}_S(r)$ and $x[\alpha(R)] \notin \delta r[alpha(R)]$: This prove is the similar to Item (i) in the proof of Equation 3 and omitted.

The containment is proved.

□

Equation 4 reveals that $\mathring{\pi}_S(r \ominus s)$ may not be contained in $(\mathring{\pi}_S(r) \ominus \mathring{\pi}_S(s))$. The following lemma gives the reason.

Lemma 4.2.
$\mathring{\pi}_S(r \ominus \delta r) = \mathring{\pi}_S(r) \ominus \mathring{\pi}_S(\delta r)$ *iff recursively* $\exists u \in r \wedge \exists v \in \delta r \wedge u[\alpha(A)] = v[\alpha(A)] \wedge \exists j \in [1...m](u[(R_j)] \ominus v[(R_j)] \neq \phi \Longrightarrow (\mathring{\pi}_{S_j^*} u[R_j^*] \ominus \mathring{\pi}_{S_j^*} v[R_j^*]) \neq \phi)$.

The proof of the Lemma is the reverse of the proof of Equation 4. Generally, $(u[R_i^*] \ominus v[R_i^*] \neq \phi) \not\Longrightarrow (\mathring{\pi}_{S_i}(u[R_i^*]) \ominus \mathring{\pi}_{S_i}(v[R_i^*]) \neq \phi)$ because the projection makes a tuple shorter. The shortened parts might be the difference of $u[R_i^*]$ and $v[R_i^*]$. Once this difference is shortened, $\mathring{\pi}_{S_i}(u[R_i^*])$ and $\mathring{\pi}_{S_i}(v[R_i^*])$ become the same. So, $(\mathring{\pi}_{S_i}(u[R_i^*]) \ominus \mathring{\pi}_{S_i}(v[R_i^*]) \neq \phi)$ may not be true. When the condition in the lemma is true, the equation becomes true.

The next example shows the importance of the condition in the lemma.

Example 4.1. Let $R = \{A, B^* : \{B\}, C^* : \{C\}\}$ and $S = \{A, B^* : \{B\}\}$. Let r and δr be two instances over R shown in Table 14. We see that $\mathring{\pi}_S(r \ominus \delta r) \neq \mathring{\pi}_S(r) \ominus \mathring{\pi}_T(\delta r)$. This is caused by the first tuple of r and in the first tuple of δr. The order of difference and projection on the two tuples affect the result.

When the two tuples are differenced first, the result is $< a_1, \phi, \{c_1\} >$, The projection of the tuple is $< a_1, \phi >$ which is in the recomputation of $\mathring{\pi}_S(r \ominus \delta r)$.

However, when the two tuples are projected, we obtain $< a_1, \{b_1\} >$ and $< a_1, \{b_1\} >$ respectively. The difference of these two tuples results none in the result.

Based on the two lemmas given above, we propose the following implementation form of IEs for the projection operator.

Theorem 4.2. *Let S be a prime subschema of R. Let r and δr be instances over R. Then, the following two equations hold.*

$$\mathring{\pi}_S(r \oplus \delta r) = \mathring{\pi}_S(r) \oplus \mathring{\pi}_S(\delta r)$$
$$\mathring{\pi}_S(r \ominus \delta r) = \mathring{\sigma}_{\alpha(R) \notin \delta r[\alpha(R)]}(\mathring{\pi}_S(r)) \oplus \mathring{\pi}_S(\mathring{\sigma}_{\alpha(R) \in \delta r[\alpha(R)]}(r) \ominus \delta r) \quad (5)$$

Table 14: Relations for Example 4.1

A	B^*	C^*
	B	C
a_1	$\{b_1\}$	$\{c_1\}$
a_2	$\{b_1, b_2\}$	$\{c_1, c_2\}$
a_3	$\{b_2, b_3\}$	$\{c_1, c_3\}$

r

A	B^*	C^*
	B	C
a_1	$\{b_1\}$	$\{c_2\}$
a_2	$\{b_1, b_2\}$	$\{c_1, c_2\}$
a_3	$\{b_3\}$	$\{c_3\}$

δr

A	B^*
	B
a_1	$\{\phi\}$
a_3	$\{b_2\}$

$\mathring{\pi}_S(r \ominus \delta r)$

A	B^*
	B
a_3	$\{b_2\}$

$\mathring{\pi}_S(r) \ominus \mathring{\pi}_S(\delta r)$

The first equation has been proved in Lemma 4.1. We now prove the second equation. From Lemma 4.2, the none equivalence of $\mathring{\pi}_S(r \ominus \delta r)$ and $\mathring{\pi}_S(r) \ominus \mathring{\pi}_S(\delta r)$ is caused by $\alpha(R)$-overlapping tuples in r and δr. Based on this observation, in Equation 5, we delete from the old view $\mathring{\pi}_S(r \ominus \delta r)$ the tuples that are produced by key value overlapping tuples in r. We then recompute the $\alpha(R)$-overlapping tuples in r and δr by $\mathring{\pi}_S(\mathring{\sigma}_{\alpha(R) \in \delta r[\alpha(R)]}(r) \ominus \delta r)$. At last, the recomputed tuples are inserted to the view.

4.3 Incremental Equations for the Selection Operator

Incremental equations for the selection operator can not be in the standard form because of the following lemma.

Lemma 4.3. *Let r and δr be two relations over schema R. Let c be a selection condition defined with Definition 2.6. Then $\mathring{\sigma}_c(r)$ is not always contained in $\mathring{\sigma}_c(r \oplus \delta r)$ and $\mathring{\sigma}_c(r \ominus \delta r)$.*

This lemma is supported by the next example.

Example 4.2. Let r and δr be two relations given in Table 15 and $c = \{B^* \neq \phi, C^* = \phi\}$ be a selection condition. The selections of r, $r \oplus \delta r$, and $r \ominus \delta r$ are also given Table 15. Obviously, $\mathring{\sigma}_c(r)$ is not contained in $\mathring{\sigma}_c(r \oplus \delta r)$ and $\mathring{\sigma}_c(r \ominus \delta r)$.

Since it is not possible to have the standard form of IEs for the selection operator, we derive IEs in the limited standard form for the selection operator if we impose restrictions on r and δr.

Lemma 4.4. *Let r and δr be two relations over schema R. Let c be a selection condition defined with Definition 2.6. Then the following equations are true.*

(i) $\mathring{\sigma}_c(r \ominus \delta r) = \mathring{\sigma}_c(r) \ominus \mathring{\sigma}_c(\delta r)$ *iff* $\exists (u \in r \wedge v \in \delta r \wedge u[\alpha(A)] = v[\alpha(A)])$, *then recursively* $\forall i \in [1, ..., n]$ $(c_{ri}(\mathring{\sigma}_{c_i}(u[R_i^*])) \ominus \mathring{\sigma}_{c_i}(v[R_i^*])) = true \wedge c_{ri}(\mathring{\sigma}_{c_i}(u[R_i^*])) = true \wedge c_{ri}(\mathring{\sigma}_{c_i}(v[R_i^*])) = true \wedge \exists i (u[R_i^*] \ominus v[R_i^*] \neq \phi \wedge \mathring{\sigma}_{c_i}(u[R_i^*]) \ominus \mathring{\sigma}_{c_i}(v[R_i^*]) \neq \phi)$

(ii) $\mathring{\sigma}_c(r \oplus \delta r) = \mathring{\sigma}_c(r) \oplus \mathring{\sigma}_c(\delta r)$ *iff* $\exists (u \in r \wedge v \in \delta r \wedge u[\alpha(A)] = v[\alpha(A)])$, *then recursively* $\forall i \in [1, ..., n]$ $(c_{ri}(\mathring{\sigma}_{c_i}(u[R_i^*])) = true \wedge c_{ri}(\mathring{\sigma}_{c_i}(v[R_i^*])) = true)$

We now choose to prove the first equation. The proof of the second equation is similar to that of the first one.

Table 15: Relations for Example 4.2

A	B^*	C^*
	B	C
a_1	$\{b_1\}$	ϕ
a_5	$\{b_5\}$	ϕ

r

A	B^*	C^*
	B	C
a_1	$\{b_2\}$	$\{c_2\}$
a_5	$\{b_5, b_6\}$	ϕ

δr

A	B^*	C^*
	B	C
a_1	$\{b_1, b_2\}$	$\{c_2\}$
a_5	$\{b_5, b_6\}$	ϕ

$r \oplus \delta r$

A	B^*	C^*
	B	C
a_5	$\{b_1\}$	ϕ

$r \ominus \delta r$

A	B^*	C^*
	B	C
a_1	$\{b_1\}$	ϕ
a_5	$\{b_5\}$	ϕ

$\mathring{\sigma}_c(r)$

A	B^*	C^*
	B	C
a_5	$\{b_5, b_6\}$	$\{\phi\}$

$\mathring{\sigma}_c(r \oplus \delta r)$

A	B^*	C^*
	B	C
a_1	$\{b_1\}$	$\{\phi\}$

$\mathring{\sigma}_c(r \ominus \delta r)$

Proof.

Proof of Equation (i) in Lemma 4.4:

(1) Base case: when $\beta(R) = \phi$, r and δr are flat. The equation becomes $\mathring{\sigma}_{c_b}(r - \delta r) = \mathring{\sigma}_{c_b}(r) - \mathring{\sigma}_{c_b}(\delta r)$. The correctness of this equation is proved in [13].

(2) Induction: when r is not flat and $\mathring{\sigma}_{c_i}(u[R_i^*] \ominus v[R_i^*]) = \mathring{\sigma}_{c_i}(u[R_i^*]) \ominus \mathring{\sigma}_{c_i}(v[R_i^*])$ where $u \in r$ and $v \in \delta r$, we prove the equation is correct below.

(a) $\mathring{\sigma}_c(r \ominus \delta r) \subseteq \mathring{\sigma}_c(r) \ominus \mathring{\sigma}_c(\delta r)$
For $x \in \mathring{\sigma}_c(r \ominus \delta r)$, there exist a tuple x' in $(r \ominus \delta r)$ such that $c(x')$ is true. x' is computed by tuple $u \in r$ and tuple $v \in \delta r$ in one of two ways.

(1) $u[\alpha(R)] \notin \delta r[\alpha(R)]$ and $x' = u$. Thus,
$x = u[\alpha(R)](\mathring{\sigma}_{c_1}(u[R_1^*])...(\mathring{\sigma}_{c_n}(u[R_n^*])$ where $c_b(u[\alpha(R)]) = true$ and $\forall\, i, c_{ri}(\mathring{\sigma}_{c_i}(u[R_i^*])) = true$.
In RHS, the selection of u produces tuple y $(= x)$ in $\mathring{\sigma}_c(r)$:
$y = u[\alpha(R)](\mathring{\sigma}_{c_1}(u[R_1^*,])...(\mathring{\sigma}_{c_n}(u[R_n^*])$ where $c_b(u[\alpha(R)]) = true$ and $\forall\, i(c_{ri}(\mathring{\sigma}_{c_i}(u[R_i^*])) = true)$.
Since selection does not change the key value of a tuple, we have $u[\alpha(R)] \notin \delta r[\alpha(R)] \Longrightarrow y[\alpha(R)] \notin \mathring{\sigma}_c(\delta r)[\alpha(R)]$. By definition of difference, no tuple in $\mathring{\sigma}_c(\delta r)$ affects y when difference is conducted in RHS. After difference, y is still the same as x. So, x is in RHS.

(2) $u[\alpha(R)] = v[\alpha(R)]$ and x' is the difference of u and v. That is,
$x' = u[\alpha(R)](u[R_1^*] \ominus v[R_1^*])...(u[R_n^*] \ominus v[R_n^*])$ where $\exists\, i(u[R_i^*] \ominus v[R_i^*] \neq \phi)$. In this case, the selection of x' gives
$x = u[\alpha(R)](\mathring{\sigma}_{c_1}(u[R_1^*] \ominus v[R_1^*]))...(\mathring{\sigma}_{c_n}(u[R_n^*] \ominus v[R_n^*]))$
where $c_b(u[\alpha(R)]) = true$ and $\forall\, i(c_{ri}(\mathring{\sigma}_{c_i}(u[R_i^*] \ominus v[R_i^*])) = true)$ and $\exists\, i(u[R_i^*] \ominus v[R_i^*] \neq \phi)$.
By induction assumption,
$x = u[\alpha(R)](\mathring{\sigma}_{c_1}(u[R_1^*,]) \ominus \mathring{\sigma}_{c_1}(v[R_1^*]))...(\mathring{\sigma}_{c_n}(u[R_n^*]) \ominus \mathring{\sigma}_{c_n}(v[R_n^*]))$

where $c_b(u[\alpha(R)]) = true$ and $\forall i(c_{ri}(\mathring{\sigma}_{c_i}(u[R_i^*] \ominus v[R_i^*])) = true)$ and $\exists i(u[R_i^*] \ominus v[R_i^*] \neq \phi)$.
In RHS , the selection of u and v results tuple $y^u \in \mathring{\sigma}_c(r)$ and tuple $y^v \in \mathring{\sigma}_c(s)$:
$y^u = u[\alpha(R)](\mathring{\sigma}_{c_1}(u[R_1^*]))...(\mathring{\sigma}_{c_n}(u[R_n^*]))$ where $c_b(u[\alpha(R)]) = true$ and $\forall i(c_{ri}(\mathring{\sigma}_{c_i}(u[R_i^*])) = true)$.
$y^v = v[\alpha(R)](\mathring{\sigma}_{c_1}(v[R_1^*]))...(\mathring{\sigma}_{c_n}(v[R_n^*]))$ where $c_b(v[\alpha(R)]) = true$ and $\forall i(c_{ri}(\mathring{\sigma}_{c_i}(v[R_i^*])) = true)$.
Let y be the difference of y^u and y^v (because of $u[\alpha(R)] = v[\alpha(R)]$):
$y = u[\alpha(R)](\mathring{\sigma}_{c_1}(u[R_1^*]) \ominus \mathring{\sigma}_{c_1}(v[R_1^*,]))...(\mathring{\sigma}_{c_n}(u[R_n^*]) \ominus \mathring{\sigma}_{c_n}(v[R_n^*]))$
where $c_b(u[\alpha(R)]) = true$, $c_b(v[\alpha(R)]) = true$, $\forall i(c_{ri}(\mathring{\sigma}_{c_i}(u[R_i^*])) = true$ and $c_{ri}(\mathring{\sigma}_{c_i}(v[R_i^*])) = true)$, and $\exists i(\mathring{\sigma}_{c_i}(u[R_i^*]) \ominus \mathring{\sigma}_{c_i}(v[R_i^*]) \neq \phi)$.
The conditions attached to x and to y are the conditions attached tot he equation. When these conditions are true, x equals to y. Hence x is in RHS.

(b) $\mathring{\sigma}_c(r) \ominus \mathring{\sigma}_c(s) \subseteq \mathring{\sigma}_c(r \ominus s)$
This proof is the reverse of the proof of Case (a).

The equation is proved.

□

We use the next example to show the importance of the conditions attached to the equations in Lemma 4.4.

Example 4.3. Let $R = \{A, B^* : \{B\}, C^* : \{C\}\}$ and let r and δr be two instances over R and given in Table 16. Let the selection condition be $c = (A =' a_1', C = \phi)$. LHS=$\mathring{\sigma}_c(r \oplus \delta r) = \phi$; while RHS= $\mathring{\sigma}_c(r) \oplus \mathring{\sigma}_c(\delta r) \neq \phi$. The reason is that the condition of subrelations over C^* being empty is violated by r and δr: the subrelation $\{c_1\}$ in r is not ϕ.

Since the equations in Lemma 4.4 can not be applied to this example. The new view has to be recomputed.

Table 16: Relations for Example 4.3

A	B^*	C^*
	B	C
a_1	$\{b_1\}$	$\{c_1\}$

r

A	B^*	C^*
	B	C
a_1	$\{b_2\}$	ϕ

δr

A	B^*	C^*
	B	C
ϕ		

$\mathring{\sigma}_c(r \oplus \delta r)$

A	B^*	C^*
	B	C
a_1	$\{b_2\}$	ϕ

$\mathring{\sigma}_c(r) \oplus \mathring{\sigma}_c(\delta r)$

The above lemma indicates that the reason for not being able to derive a standard IE for the selection operator is that r and δr having $\alpha(R)$-overlapping tuples.

The conditions attached to the equations in Lemma 4.4 are recursive. In other words, the conditions have to be tested against subrelations on all levels. This can be very time consuming because of traversing down to deep levels and because

there is no index possible for subrelations in a relation. Furthermore, after the test, the conditions may be violated, in this case, we have to recompute to view. Those tuples that have been traversed for testing the conditions will be traversed again for the recomputation. Double traversal causes the test plus recomputation to be more expensive than just recomputation without the pre-test. To overcome the disadvantage of the performance, in the next theorem we avoid the test of the conditions by proposing IEs in the implementation form.

Theorem 4.3.

$$\mathring{\sigma}_c(r \ominus \delta r) = \mathring{\sigma}_{\alpha(R) \notin \delta r[\alpha(R)]}(\mathring{\sigma}_c(r)) \oplus \mathring{\sigma}_c(\mathring{\sigma}_{\alpha(R) \in \delta r[\alpha(R)]}(r) \ominus \delta r) \quad (6)$$

$$\mathring{\sigma}_c(r \oplus \delta r) = \mathring{\sigma}_{\alpha(R) \notin \delta r[\alpha(R)]}(\mathring{\sigma}_c(r)) \oplus \mathring{\sigma}_c(\mathring{\sigma}_{\alpha(R) \in \delta r[\alpha(R)]}(r) \oplus \delta r) \quad (7)$$

In the theorem, $\alpha(R)$-overlapping tuples are recomputed while other tuples are computed incrementally. The IEs in the theorem can applied without limitation.

4.4 Incremental Equations for the Intersection Operator

The next Lemma indicates that the IE for the intersection operator with union is in the standard form. However, the IE for the intersection operator with difference can not be in the standard form since $(r \odot s) \ominus (\delta r \odot s)$ is contained in $(r \ominus \delta r) \odot s$, but not the other way around.

Lemma 4.5. *Let R be a schema and r, δr, and s be instances over R. Then*

$$(r \ominus \delta r) \odot s = (r \odot s) \ominus (\delta r \odot s) \oplus (r \ominus \delta r) \odot s \quad (8)$$

$$(r \oplus \delta r) \odot s = (r \odot s) \oplus (\delta r \odot s) \quad (9)$$

We choose to prove Equation 8. The proof of the other equation can be achieved in a similar way.

Proof.

Proof of Equation 8: To prove the equation, we only need to prove $(r \odot s) \ominus (\delta r \odot s) \underline{\subseteq} (r \ominus \delta r) \odot s$ because of Theorem 3.1.

(1) Base case: when R is flat, the equation is true since in flat case, $(r \ominus \delta r) \odot s = (r \odot s) \ominus (\delta r \odot s)$ is proved in [13].

(2) Induction: we suppose that for $u \in r \ \wedge \ v \in \delta r \ \wedge \ w \in s$, $(u[R_i^*] \odot w[R_i^*]) \ominus (v[R_i^*] \odot w[R_i^*]) \ \underline{\subseteq} \ (u[R_i^*] \ominus v[R_i^*]) \odot w[R_i^*]$. We prove $(r \odot s) \ominus (\delta r \odot s) \ \underline{\subseteq} \ (r \ominus \delta r) \odot s$.
For $x \in ((r \odot s) \ominus (\delta r \odot s))$, there exist tuple $x^u \in (r \odot s)$ and tuple $x^v \in (\delta r \odot s)$ such that x is the difference of x^u and x^v. By the definition of the difference, x is computed in the following ways from x^u and x^v.

(a) $x^u[\alpha(R)] = x^v[\alpha(R)]$ then x is the difference of x^u and x^v.
Because x^u is the intersection of $u \in r$ and $w \in s$ while x^v is the intersection of $v \in \delta r$ and $w \in s$, i.e.,

$x^u = u[\alpha(R)](u[R_1^*] \odot w[R_1^*])...(u[R_{nr}^*] \odot w[R_{nr}^*])$ and
$x^v = v[\alpha(R)](v[R_1^*] \odot w[R_1^*])...(v[R_{nr}^*] \odot w[R_{nr}^*])$.
So x is the difference of x^u and x^v:
$x = u[\alpha(R)]((u[R_1^*] \odot w[R_1^*]) \ominus (v[R_1^*] \odot w[R_1^*]))...((u[R_{nr}^*] \odot w[R_{nr}^*]) \ominus (v[R_{nr}^*] \odot w[R_{nr}^*]))$
where $(u[(R_i^*)] \odot w[R_i^*]) \ominus (v[R_i^*] \odot w[R_i^*]) \neq \phi$.
By induction assumption,
$x = u[\alpha(R)]((u[R_1^*] \ominus v[R_1^*]) \odot w[R_1^*])...((u[R_{nr}^*] \ominus v[R_{nr}^*]) \odot w[R_{nr}^*])$
where $(u[R_i^*] \odot w[R_i^*]) \ominus (v[R_i^*] \odot w[[(R_i^*)]]) \neq \phi$
In RHS, since $u[\alpha(R)] = v[\alpha(R)]$, the difference of u and v ($u \neq v$, otherwise x does not exist) in $(r \ominus s)$, denoted y', is
$y' = u[\alpha(R)](u[R_1^*] \ominus v[R_1^*])...(u[R_{nr}^*] \ominus v[R_{nr}^*])$
where $(u[R_i^*] \ominus v[R_i^*]) \neq \phi$
By intersecting y' and w ($u[\alpha(R)] = v[\alpha(R)] = w[\alpha(R)]$), we have a tuple y in RHS as
$y = u[\alpha(R)]((u[R_1^*] \ominus v[R_1^*]) \odot w[R_1^*]))...((u[R_{nr}^*] \ominus v[R_{nr}^*]) \odot w[R_{nr}^*]))$
where $(u[R_i^*] \ominus v[[(R_i^*)]]) \neq \phi$
Because $(u[R_i^*] \odot w[R_i^*]) \ominus (v[R_i^*] \odot w[R_i^*]) \neq \phi \Longrightarrow ((u[R_i^*] \ominus v[R_i^*]) \odot w[R_i^*]) \neq \phi \Longrightarrow (u[(R_i^*)] \ominus v[R_i^*]) \neq \phi$ (not vice versa), x equals to y or tuple-contained in y. Therefore x is in RHS.

(b) $x^u[\alpha(R)] \notin (\delta r \odot s)[\alpha(R)]$: then $x = x^u$.
x^u is computed by intersecting $u \in r$ and $w \in s$. That is,
$x = x^u = u[\alpha(R)](u[R_1^*] \odot w[R_1^*])...(u[R_{nr}^*] \odot w[R_{nr}^*])$.
In RHS, $u[\alpha(R)] \notin \delta r[\alpha(R)]$. This can be proved by the inverse method. Suppose there is tuple $v \in \delta r$ such that $v[\alpha(R)] = u[\alpha(R)]$, the intersection of v and w ($v[\alpha(R)] = w[\alpha(R)]$) result in a tuple $x^v \in (\delta r \odot s)$ such that $x^v[\alpha(R)] = u[\alpha(R)] = x^u[\alpha(R)]$. This is a contradiction to the condition of $x^u[\alpha(R)] \notin (\delta r \odot s)[\alpha(R)]$. Thus, $u[\alpha(R)] \notin \delta r[\alpha(R)]$.
From this point, we conclude that there is not tuple in δr affecting u during the difference of $(r \ominus s)$. So the intersection of u and w is the same as x. Consequently, x is in RHS.

This proof of $((r \odot s) \ominus (\delta r \odot s)) \subseteq ((r \ominus s) \odot t)$ is done.

The equation is proved. □

If restrictions are placed on relations r and δr, it is possible to derive IEs in the limited standard form for the intersection operator as shown in the following result.

Lemma 4.6. $(r \ominus \delta r) \odot s = (r \odot s) \ominus (\delta r \odot s)$, *if recursively*
$\exists u \in r \ \wedge \ \exists v \in \delta r \ \wedge \ \exists w \in s \ \wedge \ u[\alpha(R)] = v[\alpha(R)] = w[\alpha(R)] \ \wedge$
$\exists i(u[(R_i)] \ominus v[(R_i)] \neq \phi \ \wedge \ (u[(R_i)] \odot w[(R_i)]) \ominus (v[(R_i)] \odot w[(R_i)]) \neq \phi)$.

The condition attached to the lemma is extracted from the proof of Equation 8. We now use Example 4.4 to show that a standard IE is not valid unless the relations satisfy the condition of Lemma 4.6.

Example 4.4. Let r, δr and s be relations given in Table 17. After recomputation and incremental computation, $(r \ominus \delta r) \odot s \neq (r \odot s) \ominus (\delta r \odot s)$. This is because the first tuple in r, δr and s violates the condition.

Table 17: Relations for Example 4.4

A	B^*
	B
a_1	$\{b_1, b_2\}$
a_2	$\{b_2, b_3\}$
a_3	$\{b_4\}$
a_4	$\{b_5\}$

r

A	B^*
	B
a_1	$\{b_1, b_3\}$
a_2	$\{b_1, b_2\}$
a_3	$\{b_3, b_4\}$

δr

A	B^*
	B
a_1	$\{b_1\}$
a_2	$\{b_2, b_3\}$
a_3	$\{b_4\}$
a_4	$\{b_5\}$

s

A	B^*
	B
a_1	ϕ
a_2	$\{b_3\}$
a_4	$\{b_5\}$

$(r \ominus \delta r) \odot s$

A	B^*
	B
a_2	$\{b_3\}$
a_4	$\{b_5\}$

$(r \odot s) \ominus (\delta r \odot s)$

A closer inspection of this example and the previous results indicates that the reason for not being able to derive a standard IE for the intersection operator is caused by $\alpha(R)$-overlapping tuples in r and δr. If we treat these tuples separately then we can derive IEs in the implementation form that is more efficient than recomputation.

Theorem 4.4. *Let R be a schema and r, δr, and s be instances over R. Then*

$$(r \ominus \delta r) \odot s = \mathring{\sigma}_{\alpha(R) \notin \delta r[\alpha(R)]}(r \odot s) \oplus ((\mathring{\sigma}_{\alpha(R) \in \delta r[\alpha(R)]}(r)) \ominus \delta r) \odot s \quad (10)$$

$$(r \oplus \delta r) \odot t = (r \odot s) \oplus (\delta r \odot s) \quad (11)$$

The next example shows the usage of Equation 10. The example also shows that Equation 10 is more efficient than recomputation.

Example 4.5. Let r, δr and s be relations given in Table 17. The incremental computation using Equation 10 is given in Table 18. We see that $z_1 \oplus z_2$ is the same as recomputation $(r \ominus \delta r) \odot t$ in Table 17. We note that we did not recompute the intersection of tuple $< a_4, \{b_5\} >$ in r and s. This is where Equation 10 is cheaper than recomputation.

Table 18: Use of Equation 10

A	B^*
	B
a_1	$\{b_1\}$
a_2	$\{b_2, b_3\}$
a_3	$\{b_4\}$
a_4	$\{b_5\}$

$r \odot s$

A
a_1
a_2
a_3

$\delta r[A]$

A	B^*
	B
a_4	$\{b_5\}$

z_1

A	B^*
	B
a_1	ϕ
a_2	$\{b_3\}$

z_2

A	B^*
	B
a_1	ϕ
a_2	$\{b_3\}$
a_4	$\{b_5\}$

$z_1 \oplus z_2$

$z_1 = \mathring{\sigma}_{\alpha(R) \notin \delta r[\alpha(R)]}(r \odot s) \quad z_2 = ((\mathring{\sigma}_{\alpha(R) \in \delta r[\alpha(R)]}(r)) \ominus \delta r) \odot s$

4.5 Incremental Equations for Join Operator

Theorem 4.5. *Let R and S be joinable schemas. Let r and δr be instances over R and s be an instance over S. The following two equations hold.*

$$(r \ominus \delta r) \overset{\diamond}{\bowtie} s = (r \overset{\diamond}{\bowtie} s) \ominus (\delta r \overset{\diamond}{\bowtie} s) \oplus (r \ominus \delta r) \overset{\diamond}{\bowtie} s \tag{12}$$

$$(r \oplus \delta r) \overset{\diamond}{\bowtie} s = (r \overset{\diamond}{\bowtie} s) \oplus (\delta r \overset{\diamond}{\bowtie} s) \tag{13}$$

We choose to prove the first equation. The proof of the second equation is similar to that of the first one.

Proof.

Proof of Equation 12: To prove the equation, we only need to prove $(r \overset{\diamond}{\bowtie} s) \ominus (\delta r \overset{\diamond}{\bowtie} s) \subseteq (r \ominus \delta r) \overset{\diamond}{\bowtie} s$ because of Theorem 3.1.

Let $R = \alpha(R)\{R^*_1, ..., R^*_{nr}\}$ and $S = \alpha(R)\{S^*_1, ..., S^*_{ns}\}$. Let r and δr be relations on R and s be a relation on S. Let $T = \alpha(R)\{T^*_1, ..., T^*_{nt}\}$ be joined schema of R and S.

(1) Base case: when $R = S = T = \alpha(R)$, i.e. all schemas are flat and the same, $(r \overset{\diamond}{\bowtie} s) \ominus (\delta r \overset{\diamond}{\bowtie}_r s) = (r \ominus \delta r) \overset{\diamond}{\bowtie} s$. This is because when relations are flat, join operation is equivalent to set intersection operation. The equivalence is proved in [13].

(2) Induction: we assume that for $u \in r \wedge v \in \delta r \wedge w \in s$, $(u[R^*_j] \overset{\diamond}{\bowtie} w[(S_k)]) \ominus (v[R^*_j] \overset{\diamond}{\bowtie} w[(S_k)]) \subseteq (u[R^*_j] \ominus v[R^*_j]) \overset{\diamond}{\bowtie} w[(S_k)]$. We prove $(r \overset{\diamond}{\bowtie} s) \ominus (\delta r \overset{\diamond}{\bowtie} s) \subseteq (r \ominus \delta r) \overset{\diamond}{\bowtie} s$.
For $x \in ((r \overset{\diamond}{\bowtie} s) \ominus (\delta r \overset{\diamond}{\bowtie} s))$ in LHS, there exist $x^u \in (r \overset{\diamond}{\bowtie} s)$ and $x^v \in (\delta r \overset{\diamond}{\bowtie} s)$ such that x is the difference of x^u and x^v. By the definition difference, we have

$$\begin{aligned}
&(r \overset{\diamond}{\bowtie} t) \ominus (s \overset{\diamond}{\bowtie} t)\\
&= \{x | x[\alpha(R)] = x^u[\alpha(R)] = x^v[\alpha(R)] \ \wedge \ \forall i (x[T^*_i] = x^u[T^*_i] \ominus x^v[T^*_i]) \wedge\\
&\qquad \exists i (x[T^*_i] \neq \phi) \text{ or}\\
&\qquad x[\alpha(R)] = x^u[\alpha(R)] \neq \forall x^v[\alpha(R)] \ \wedge \ x = x^u \qquad \}
\end{aligned}$$

where by the definition of join,

$$\begin{aligned}
x^u \in \{x^u | &\exists u \in r \wedge \exists w \in s \wedge x^u[\alpha(R)] = u[\alpha(R)] = w[\alpha(R)] \wedge \forall i\\
&(x^u[T^*_i] = u[R^*_j] \overset{\diamond}{\bowtie} w[S^*_k], \text{ if } R_j \subseteq^p T_i \wedge S_k \subseteq^p T_i \text{ or}\\
&x^u[T^*_i] = u[R^*_j], \text{ if } R_j \subseteq^p T_i \wedge \nexists S_k \subset T_i \text{ or}\\
&x^u[T^*_i] = w[S^*_k], \text{ if } \nexists R_j \subset T_i \wedge S_k \subseteq^p T_i) \qquad \} \text{ and}\\
x^v \in \{x^v | &\exists v \in \delta r \wedge \exists w \in s \wedge x^v[\alpha(R)] = v[\alpha(R)] = w[\alpha(R)] \wedge \forall i\\
&(x^v[T^*_i] = v[R^*_j] \overset{\diamond}{\bowtie} w[S^*_k], \text{ if } R_j \subseteq^p T_i \wedge S_k \subseteq^p T_i \text{ or}\\
&x^v[T^*_i] = v[R^*_j], \text{ if } R_j \subseteq^p T_i \wedge \nexists S_k \subset T_i \text{ or}\\
&x^v[T^*_i] = w[S^*_k], \text{ if } \nexists R_j \subset T_i \wedge S_k \subseteq^p T_i) \qquad \}
\end{aligned}$$

We replace x^u and x^v with their definition in the difference. Because the schema of x_u and x_v are the same, each case in x^u matches that in x^v, and does not match any of the other two cases. For this reason, we first consider the first case of x_u and the first case of x_v in the first case of the difference.

We then consider the second and third cases of x_u and x_v respectively in the first case of the difference. After that, we consider each case of x_u and x_v in the second case of the difference. From this procedure, we get the following expression. In the expression, for the simplicity, we do not write the schema conditions explicitly below.

$$
\begin{aligned}
&(r \bowtie s) \ominus (\delta r \bowtie s) \\
&= \{x \mid (\exists u \in r \ \wedge \ \exists v \in \delta r \ \wedge \ \exists w \in s \ \wedge \\
&\qquad x[\alpha(R)] = u[\alpha(R)] = v[\alpha(R)] = w[\alpha(R)] \ \wedge \\
&\qquad \forall i (x[T_i^*] = (u[R_j^*] \bowtie w[S_k^*]) \ominus (v[R_j^*] \bowtie w[S_k^*]) \text{ or} \\
&\qquad\quad x[T_i^*] = u[R_j^*] \ominus v[R_j^*] \text{ or} \\
&\qquad\quad x[T_i^*] = w[S_k^*] \ominus w[S_k^*]) \ \wedge \\
&\qquad \exists i (x[T_i^*] \neq \phi) \qquad\qquad \text{or} \\
&\quad (\exists u \in r \ \wedge \ \forall v \in \delta r \ \wedge \ \exists w \in s \ \wedge \\
&\qquad x[\alpha(R)] = u[\alpha(R)] = w[\alpha(R)] \neq v[\alpha(R)] \ \wedge \\
&\qquad \forall i (x[T_i^*] = (u[R_j^*] \bowtie w[S_k^*]) \text{ or} \\
&\qquad\quad x[T_i^*] = u[R_j^*] \text{ or} \\
&\qquad\quad x[T_i^*] = w[S_k^*]) \qquad\qquad \}
\end{aligned}
$$

Now we consider $(r \ominus \delta r) \bowtie s$ in RHS. Let $y \in (r \ominus \delta r) \bowtie s$ and $y^u \in (r \ominus \delta r)$. By the definition of join, we have

$$
\begin{aligned}
&(r \ominus \delta r) \bowtie s \\
&= \{y \mid y[\alpha(R)] = y^u[\alpha(R)] = w[\alpha(R)] \ \wedge \\
&\quad \forall i (y[T_i^*] = y^u[R_j^*] \bowtie w[S_k^*], \text{ if } R_j \subseteq^p T_i \ \wedge \ S_k \subseteq^p T_i \text{ or} \\
&\qquad y[T_i^*] = y^u[R_j^*], \text{ if } R_j \subseteq^p T_i \ \wedge \ \not\exists S_k \subset T_i \text{ or} \\
&\qquad y[T_i^*] = w[S_k^*], \text{ if } \not\exists R_j \subset T_i \ \wedge \ S_k \subseteq^p T_i \qquad) \qquad \}
\end{aligned}
$$

where by the definition of difference,

$$
\begin{aligned}
y^u \in \{y^u \mid &\exists u \in r \ \wedge \ \exists v \in \delta r \ \wedge \ y^u[\alpha(R)] = u[\alpha(R)] = v[\alpha(R)] \ \wedge \\
&\forall j (y^u[R_j^*] = u[R_j^*] \ominus v[R_j^*]) \ \wedge \ \exists j (u[R_j^*] \ominus v[R_j^*] \neq \phi) \text{ or} \\
&\exists u \in r \ \wedge \ \forall v \in \delta r \ \wedge \ y^u[\alpha(R)] = u[\alpha(R)] \neq v[\alpha(R)] \ \wedge \ y^u = u \ \}
\end{aligned}
$$

Replace y^u with its definition in the RHS,

$$
\begin{aligned}
&(r \ominus \delta r) \bowtie s \\
&= \{y \mid \exists u \in r \wedge \exists v \in \delta r \ \wedge \ \exists w \in s \wedge \\
&\qquad y[\alpha(R)] = u[\alpha(R)] = v[\alpha(R)] = w[\alpha(R)] \ \wedge \\
&\qquad \forall i (y[T_i^*] = (u[R_j^*] \ominus v[R_j^*]) \bowtie w[S_k^*] \text{ or} \\
&\qquad\quad y[T_i^*] = u[R_j^*] \ominus v[R_j^*] \text{ or} \\
&\qquad\quad y[T_i^*] = w[S_k^*] \ ) \\
&\qquad \exists j (u[R_j^*] \ominus v[R_j^*] \neq \phi) \text{ or} \\
&\quad \exists u \in r \ \wedge \ \forall v \in \delta r \ \wedge \ \exists w \in s \wedge \\
&\qquad y[\alpha(R)] = u[\alpha(R)] = w[\alpha(R)] \neq v[\alpha(R)] \ \wedge \\
&\qquad \forall i (y[T_i^*] = u[R_j^*] \bowtie w[S_k^*] \text{ or} \\
&\qquad\quad y[T_i^*] = u[R_j^*] \text{ or} \\
&\qquad\quad y[T_i^*] = w[S_k^*] \ ) \qquad\qquad \}
\end{aligned}
$$

Now we compare x and y. The second case of x equals to the second case

of y with conditions. With the induction assumption, the first case of x equals to the first case of y with the condition that $\exists i(x[T_i^*] \neq \phi) \iff \exists j(u[R_j^*] \ominus v[R_j^*] \neq \phi)$.
We now prove that $\exists i(x[T_i^*] \neq \phi) \implies \exists j(u[R_j^*] \ominus v[R_j^*] \neq \phi)$ but not vice versa. In the first case of x, $x[T_i^*]$ is computed in two ways (note that $w[S_k^*] \ominus w[S_k^*]$ always results ϕ while we are discussing not being ϕ).

- $x[T_i^*] = u[R_j^*] \ominus v[R_j^*] \neq \phi$. This is equivalent to $\exists\,(u[R_j^*] \ominus v[R_j^*]) \neq \phi$ of y side.
- $x[T_i^*] = (u[R_j^*] \bowtie w[S_k^*]) \ominus (v[R_j^*] \bowtie w[S_k^*]) \neq \phi$. By the induction assumption, we have $x[T_i^*] = (u[R_j^*] \ominus v[R_j^*]) \bowtie w[S_k^*] \neq \phi$ By the definition of join, $(u[R_j^*] \ominus v[R_j^*]) \bowtie w[S_k^*] \neq \phi \implies (u[R_j^*] \ominus v[R_j^*]) \neq \phi$.
  However, from $(u[R_j^*] \ominus v[R_j^*]) \neq \phi$ we can not get $(u[R_j^*] \ominus v[R_j^*]) \bowtie w[S_k^*] \neq \phi$ because the join of two non-empty sets can be an empty set. Consequently, $(u[R_j^*] \ominus v[R_j^*]) \neq \phi \not\Longrightarrow (u[R_j^*] \bowtie w[S_k^*]) \ominus (v[R_j^*] \bowtie w[S_k^*]) \neq \phi$.

Because the condition of x can lead to the condition of y to be true, any x in LHS is contained or tuple-contained in RHS. However, the condition of y can not always lead to the condition of x to be true, y is not always included in RHS.
In summary, we proved that $(r \bowtie s) \ominus (\delta r \bowtie s) \mathbin{\underline{\subset}} (r \ominus \delta r) \bowtie s$.

□

Equation 12 can be simplified if restrictions are placed on the update δr.

Lemma 4.7. *Let R and S be two joinable schemas. Let r and δr be instances over R and s be an instance over S. The following equation holds.*
$(r \ominus \delta r) \bowtie s = (r \bowtie s) \ominus (\delta r \bowtie s)$ *iff r and δr are $\alpha(R)$-disjoint.*

The proof of the lemma is similar to the proof of Equation 12. We now give an example to show the importance of the condition in the lemma.

Example 4.6. Let r, δr, and s be three relations given in Table 19. The recomputation $(r \ominus \delta r) \bowtie s$ and the incremental computation $(r \bowtie s) \ominus (\delta r \bowtie s)$ of the join operator are also included in the table. Because r and δr have $\alpha(R)$-overlapping tuples, the recomputation contains more tuple than incremental computation does.

Consider the performance and implementation, as we analyzed in Subsection 4.3, we give the following IEs in implementatation form. In the theorem, $\alpha(R)$-overlapping tuples in r and δr are recomputed while other tuples in δr are incrementally computed.

Theorem 4.6. *Let R and S be two joinable schemas. Let r and δr be instances over R and s be an instance over S. The following two equations hold.*

$$(r \ominus \delta r) \bowtie s = \mathring{\sigma}_{\alpha(R) \notin \delta r[\alpha(R)]}(r \bowtie s) \oplus ((\mathring{\sigma}_{\alpha(R) \in \delta r[\alpha(R)]}(r) \ominus \delta r) \bowtie s) \quad (14)$$

$$(r \oplus \delta r) \bowtie s = (r \bowtie s) \oplus (\delta r \bowtie s) \quad (15)$$

Table 19: Relations for Example 4.6

A	B^*	C^*
	B	C
a_1	$\{b_1\}$	$\{c_1\}$
a_2	$\{b_1\}$	$\{c_2, c_3\}$
a_3	$\{b_1, b_2\}$	$\{c_1\}$

r

A	B^*	C^*
	B	C
a_1	$\{b_1\}$	$\{c_1\}$
a_2	$\{b_1\}$	$\{c_3\}$
a_3	$\{b_2, b_3\}$	$\{c_1\}$

δr

A	C^*	D^*
	C	C
a_1	$\{c_1\}$	$\{d_1\}$
a_2	$\{c_1\}$	$\{d_2\}$
a_3	$\{c_1\}$	$\{d_3\}$

s

A	B^*	C^*	D^*
	B	C	D
a_2	ϕ	ϕ	$\{d_2\}$
a_3	$\{b_1\}$	ϕ	$\{d_3\}$

$(r \ominus \delta r) \hat{\bowtie} s$

A	B^*	C^*	D^*
	B	C	D
a_3	$\{b_1\}$	ϕ	ϕ

$(r \hat{\bowtie} s) \ominus (\delta r \hat{\bowtie} s)$

4.6 Discussion

In this subsection, we highlight some of the important features of the IEs derived in the last section and discuss the differences between incremental expressions for flat relations and for nested relations.

Firstly, we note that some (but not all) of the incremental expressions are in the standard form since the expressions do not involve recomputing the new view. The performance gain of this type of IEs is obvious. This is the case where the update is an insertion and the operators are expansion, intersection, projection, or join. However, the IEs for some operators, e.g. Equation 8, involve recomputation. These types of IEs do not avoid view recomputation unless restrictions are placed upon the update. A similar situation occurs with the flat relational projection operator where the incremental expression involves view recomputation [13]. This highlights the fact that some views are impossible to maintain efficiently if the only information available is the view itself. This has lead to the development of other techniques which use counts [6] or auxiliary relations [19] to improve efficiency by avoiding view recomputation. Similarly, one expects that views involving nested operators can be more efficiently maintained if more information than just the view is stored.

In comparing the equations derived in the last section and those in [13] for the flat relational model, one notes that not only are the equations in this paper generally more complex but also the symmetry shown in the equations of [13] is absent in the expressions of the nested operators. For example, in flat relations the incremental expressions for selections and joins are similar for both insertions and deletions and are computed as $\hat{\sigma}_c(r \pm s) = \hat{\sigma}_c(r) \pm \hat{\sigma}_c(s)$ and $(r \pm s) \hat{\bowtie} t = r \hat{\bowtie} t \pm s \hat{\bowtie} t$ respectively. This symmetry is absent in the equations derived in Section 4.

Apart from the difference of patterns of IEs, the containment and disjointedness properties used in deriving IEs for flat relations and those used for nested relations are different. The containment and disjointedness for flat relations are defined on set semantics. A tuple contained in a set means that the tuple is a member of the set. In nested relations, however, a tuple is contained in a nested relation does not

mean that the tuple exists in the relation. The tuple may be contained in a tuple of the relation. Tuple containment becomes a core concept in the containment for nested relations. Similarly, the disjointedness for nested relations is built on the basis that two tuples are disjoint if one tuple has a subrelation that is disjoint from the corresponding subrelation of the other tuple.

Compared with the equations for flat relations, our equations in the implementation form incrementally compute the view update to the maximum extent and avoid full view recomputation by recomputing only key attribute overlapping tuples. The performance gain from the implementation form is obvious. This is because in the implementation form, we make full use of indexes on the top level to select key attribute overlapping tuples. We avoid to test the complex conditions of IEs in the limited standard form and avoid full view recomputation.

5 Incremental Maintenance of Views with Complex Queries

In this section, we present a technique for the incremental maintenance of PNF views expressed as complex queries using the incremental equations derived previously. Our technique can handle both insertions and deletions. However, for simplicity of exposition we assume that the update is a single insertion to a base relation.

Our technique is firstly to represent the query expression for the view as an expression or operator tree. In this representation, the leaves of the tree are the base relations, the interior nodes are the query operators and the root of the tree represents the final view. Our technique then computes the change to the view in a 'bottom' up fashion starting with the changes to the leaf nodes and then propagating the changes upwards in the expression tree using the incremental equations derived previously. To be more precise, we can express the technique in the following algorithm (ir_i represents the intermediate relation corresponding to node i in the tree, $ir_{i'}$ and $ir_{i''}$ are the intermediate relations corresponding to the child nodes of node i).

Algorithm 5.1. (Maintaining views with complex queries)
Input: the operator tree and the insertions δr to r
Output: update to the view
Do: For each node i do
  if $\delta ir_{i'}$ or $\delta ir_{i''}$ is non-empty, then
    compute δir_i according to IEs derived in previous sections.
  endif;
endfor;

The use of the algorithm is illustrated by the next example.

Example 5.1. Let relation *pstud* be defined in Table 1. Let relation *subjLect* be defined in Table 20. We define a query to list

- the name of a student;
- the subject taken by the student in 1998 if the student gets all marks over 69 for every test;
- the lectures for the taken subject.

This query can be expressed as

$$View = \mathring{\pi}_{\{Name, Subj^*:\{sjName, Year, Lect^*:\{Lect\}\}\}}((\mathring{\sigma}_{Subj^*:(Year=98 \ \wedge \ Marks:(Mark>79)\neq\phi)\neq\phi}(pstud)) \mathring{\bowtie} subjLect))$$

Table 21 shows the instance of the view.

Table 20: Relation *subjLect*

Name	*Subjs**		
	sjName	*Year*	*Lects**
			Lect
Jack	*DB*	98	{*Ben*, *Tom*}
	DB	96	{*Kaven*}
John	*DB*	98	{*Ben*, *Tom*}
	DB	96	{*Kaven*}

Table 21: The materialized view *View* before update

Name	*Subjs**		
	sjName	*Year*	*Lects**
			Lect
Jack	{ *DB*	98	{*Ben*, *Tom*}}

We now update relation *pstud*. The update to *pstud* is an insertion $\Delta pstud$ given in Table 22. The update is propagated through the following steps.

- $\Delta V_3 = \mathring{\sigma}_{Subj^*:(Year=98 \ \wedge \ Marks:(Mark>79)\neq\phi)\neq\phi}(\Delta pstud)$
  $=\{< Sean, \{< DB,\ 98,\ \{< exam,\ 90 >\} >\}, \phi >\}$
- $\Delta V_2 = \Delta V_3 \mathring{\bowtie} subjLect$
  $=\{< Sean, \{< DB,\ 98,\ \{< exam,\ 90 >\}, \{< Ben >, < Tom >\} >\}, \phi >\}$
- $\Delta V_1 = \mathring{\pi}_{\{Name, Subj^*:\{sjName, Year, Lect^*:\{Lect\}\}\}}(\Delta V_2)$
  $=\{< Sean, \{< DB,\ 98,\ \{< Ben >, < Tom >\} >\} >\}$
- $View = View \oplus \Delta V_1$ is given in Table 23.

In the above procedure, we only computed the view update having the tuple of 'Sean'. We did not compute the tuple 'Jack' that had been in the old view. When tuples in the old view are numerous, our way of maintaining the view can have better performance than recomputation.

Table 22: An update Δr to r

$Name$	$Subjs^*$				$Tels^*$
	$sjName$	$Year$	$Marks^*$		Tel
			$testName$	$Mark$	
$Sean$	$\{DB$	98	$\{exam$	$90\}\ \}$	ϕ

Table 23: The materialized view $View$ after update

$Name$	$Subjs^*$		
	$sjName$	$Year$	$Lects^*$
			$Lect$
$Jack$	$\{\ DB$	98	$\{Ben, Tom\}\}$
$Sean$	$\{\ DB$	98	$\{Ben, Tom\}\}$

6 Implementation and Performance Analysis of IEs

In this section, we present the results of our performance analysis of the IEs we derived. A detailed description of implementation can be found in [12].

6.1 The database

We employed a university database for the implementation. It contains five relations. The schema of each relation is described in Figure 24. The schema *Stud* has been described in Example 1. The schema *Teach* describes the lecturer names ($lcName$) for each subject in a year. *Lect* is a schema modeling the information of lecturers. *Test* is the schema to describe the test details for a subject in a year. The last schema, *Hobby*, describes hobbies of a student.

Table 24: Schemas of the university database

$Name$	$Subjs^*$				$Addrs^*$
	$sjName$	$Year$	$Marks^*$		$Addr$
			$TestName$	$Mark$	

schema *Stud* of relation *pstud*

$sjName$	$Year$	$lcNames^*$
		$lcName$

schema *Teach* of relation *teach*

$lcName$	$Salary$	$Speciality$

schema *Lect* of relation *lc*

$sjName$	$Year$	$TestNames^*$	
		$TestName$	$Description$

schema *Test* of relation *test*

$Name$	$Hobbies^*$
	$Hobby$

schema *Hobby* of relation *hobby*

The implementation was performed on a Pentium 166 PC computer with two hard disks, 96 MB of memory, and Microsoft Windows NT 4.0 operation system.

The database management system used was the Informix Database Server with Universal Data Operation (IDS/UDO) version 9.14. Clients interface and programs are connected to the server by TCP/IP *loop-back* connection [9], which is the only option for Windows NT platform. The query language we use for the implementation is the OR-SQL proposed in [17].

6.2 The Cost Model

The cost model for the performance analysis involves the costs of view creation, incremental maintenance, and recomputation. Each cost is the time for computing an item in an IE. We use the IE for the join operator with a deletion update as an example to show the relationship between the costs in the cost model and the items of the equations following.

$$\underbrace{(r \ominus \delta r) \mathbin{\mathring{\bowtie}} s}_{c_{rec}} = \underbrace{\underbrace{\mathring{\sigma}_{\alpha(R) \notin \delta r[\alpha(R)]}(\overbrace{r \mathring{\bowtie} s}^{c_{cre}})}_{c_{del}} \underbrace{\oplus}_{c_{ins}} \underbrace{(\mathring{\sigma}_{\alpha(R) \in \delta r[\alpha(R)]}(r)}_{c_{ovl}} \underbrace{\ominus}_{c_{cmb}} \delta r) \underbrace{\mathring{\bowtie}}_{c_{inc}} s}_{c_{mtn}} \tag{16}$$

We now detail each cost.

- c_{cre} is the time for creating the materialized view $r \mathring{\bowtie} s$.
- c_{rec}, on the left hand side, is the time for recomputing the view when an update happens to a deriving relation of the view.
- c_{mtn}' on the right hand side, is the time for incrementally maintaining the view using right hand side of incremental equations when an update happens to a deriving relation of the view. This time consists of the following components.
  - c_{del} is the time for deleting from the old view the tuples derived from $\alpha(R)$-overlapping tuples in r.
  - c_{ins} is the time for inserting the tuples of the view update into the view. Since the tuples in the view update are $\alpha(R)$-disjoint with the view because of the select operation against r, this insertion in fact is the set operation.
  - c_{ovl} is the time for selecting $\alpha(R)$-overlapping tuples from r, the relation that is being updated.
  - c_{cmb} labels the time to conduct PNF union or difference between $\alpha(R)$-overlapping tuples of r and δr.
  - c_{inc} denotes the time for computing the view update using the operator that defines the view (e.g., $\mathring{\bowtie}$).

  After defining all the costs, the total maintenance time is given by:

$$c_{mtn} = c_{del} + c_{ins} + c_{ovl} + c_{cmb} + c_{inc}$$

  where costs of c_{del}, c_{ins}, c_{ovl}, and c_{cmb} are not operator specific.

With this cost model, a typical performance analysis diagram is like the one shown in Figure 2. The horizontal axis indicates the different update sizes while the vertical axis indicates the relative time to view creation. There are three lines in the diagram. One is labeled by 'rec' and shows the relative time of view recomputation: c_{rec}/c_{cre}. It goes downward from top-left corner when updates are deletions. When the update size reaches 50% of the original size, it should come down to 50% along the vertical axes.

The second line is labeled by 'inc' and shows the relative time of c_{inc}/c_{cre}. It goes up from the lower-left corner. This line is drawn by using updates that do not have A-overlapping tuples with r. Because there are no A-overlapping tuples in the update, no recomputation and no deletion from the old view are needed for the incremental maintenance. Therefore, It is an ideal line for incremental maintenance. When the size of the update reaches 50%, this line will cross with 'rec' at 50% of the vertical axis. This line and the location of the cross serve to check the correct of the implementation programs.

The third line labeled by 'mtn' is the relative time for general incremental maintenance: c_{mtn}/c_{cre}. It goes up from the lower-left corner. The intersection of the two lines 'rec' and 'mtn' being located at over 50% of the vertical axis and less than 50% of the horizontal axis. The horizontal coordinate of the the intersection point is called the *maintenance limit.* It is the size of an update with which the time of incremental maintenance is equivalent to the time of view recomputation.

The 'mtn' line in the figure is the worst case where all tuples in δr are $\alpha(R)$-overlapping with r and produce tuples in the view update to be inserted into the old view. Lines 'mtn' and 'inc' are the minimum and maximum boundaries for the incremental maintenance. The actual maintenance limit, depending on the update type and $\alpha(R)$-overlapping property, falls within the boundaries.

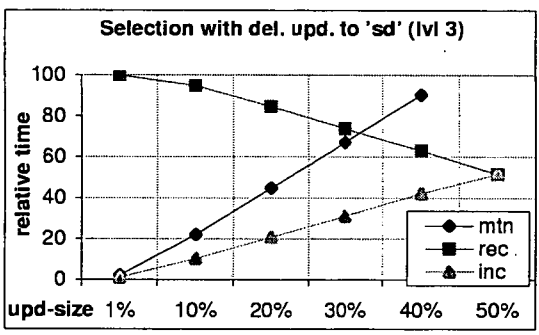

Figure 2: The performance analysis for selection on 3rd level

6.3 Maintenance Limit

In this section, we analyze the maintenance limit for the selection operator and the join operator.

6.3.1 Selection Operator

To study the maintenance limit for the selection operator, we simulate a query of listing all tuples of table *stud* if a student has at least one good mark (≥ 90) for at least one subject. The query is

```
Q_s: SELECT * FROM stud s WHERE EXISTS
       (SELECT * FROM table(s.Subjs*) j WHERE EXISTS
          (SELECT * FROM table(j.Marks*) WHERE Mark≥ 90) );
```

The selection condition $Mark \geq 90$ in the query is set up on third level of relation *stud* with the selectivity being 10%.

The view defined with this query is maintained using the right hand side of Equation 6.

The performance analysis diagram has been given Figure 2. From the diagram, we see that the maintenance limit is about 32%.

We also analyzed the maintenance limits for the cases where selection conditions are on the first level and the second level respectively. The maintenance limits for selection condition on all levels are listed in Table 25.

Table 25: Maintenance limits for selection

condition level	1st	2nd	3rd
limits(%)	18	40.5	32

6.3.2 Join Operator

The query we study the maintenance limit of join operator is Q_{jn}.

```
Q_jn: SELECT * FROM stud s WHERE EXISTS
        (SELECT * FROM table(s.Subjs*) j, test t
          WHERE j.sjName=t.sjName AND j.Year=t.Year
          AND EXISTS (SELECT * FROM table(j.Marks*) a, table(t.TestNames*) b
            WHERE a.TestName=b.TestName) );
```

In this query, *test* joins *Stud* on the second level and the third level of *Stud*.

We also simulated join operations on the first level and the second. The tree presentation of the join in the three levels is given in Figure 3. The maintenance limits for the three case is given in Table 26.

Table 26: Maintenance limits for join

condition level	1st	2nd	2nd & 3rd
limits(%)	21	44	36

The data in the table is quite similar to that of Table 25. So, we omit the explanation.

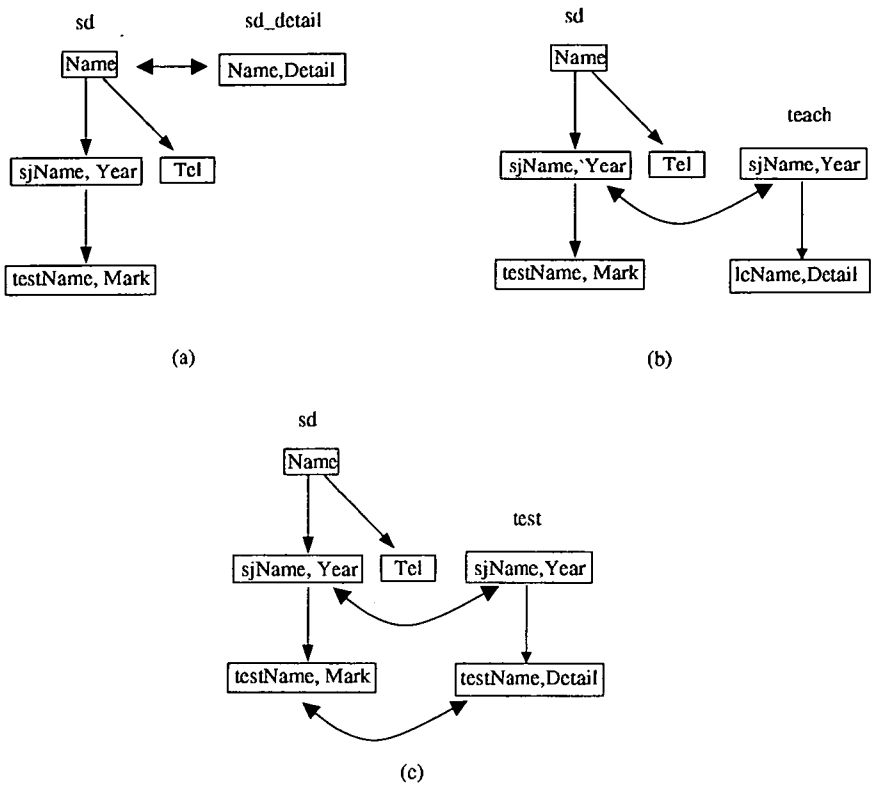

Figure 3: Joins on different levels

The above data is obtained by applying updates to table *stud*, which we call the left operand of the join. We now consider the cases where updates are applied to the right operand of the join.

Line 'inc-lv3' in Figure 6.3.2 is the case where the right operand joins *stud* on the third level. The figure shows that the incremental maintenance cost does not vary with the change of the size of updates to the right operand. This maintenance cost is almost the same as the view creation time. This is because the navigation of *stud* down to the third level consumes most of the time of the join operation. Line 'inc-lv2' and Line 'inc-lv1' of the figure indicate that as the level on which the right operand joins *stud* becomes shallower, the cost of the incremental maintenance changes toward the trend of updates to *stud*.

7 Conclusion

In this paper, we derived IEs for the operators of PNF nested relations. We derived IEs in three forms: the standard form, the limited standard form, and the imple-

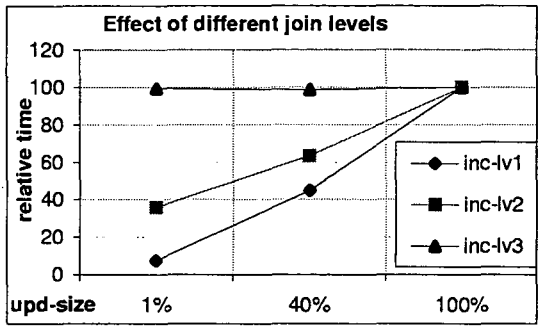

Figure 4: The performances when other joining tables are updated

mentation form. The standard form is the ideal form that we first aimed to achieve. If an IE can not be in the standard form, we proposed the limited standard form. This form aims to reveal the reason why the IE can not be standard. After the limited standard form, by considering performance of testing complex conditions, we have derived IEs in the implementation form to avoid testing such conditions.

In this paper, we also implemented IEs for the nested relations in the Informix Universal Database Server. A database with multi-level nested relations was created in the database server. We implemented the PNF operators using ESQL/C functions. With the operators and the relations in the database, views were created and the incremental equations are implemented. Afterward, the performance of each incremental equation was analyzed.

The performance analysis show that the PNF union and difference operations are the main reasons causing performance decrease of the incremental computation. Generally, the maintenance limits of the incremental equations are between 17–44%. As the number of nested levels increase, the maintenance limit decreases.

Nested relations are closely related to the new emerged semi-structured data protocol XML (eXtensible Markup Language) [1]. Because of this, the IE expressions derived in this paper have the potential to be adapted for the use in maintaining XML views. However, the adaption will not be direct because XML allows missing elements and flexible structures. This leads to null sub relations which challenge the adaption. We leave this as future research work.

References

[1] Serge Abiteboul. On views and xml. *PODS*, pages 1–9, 1999.

[2] Serge Abiteboul and Nicole Bidoit. Non first normal form relations: an algebra allowing data restructuring. *Journal of Computer and System Sciences*, 33(3):361–393, 1986.

[3] Stijn Dekeyser, Bart Kuijpers, and Jan Paredaens. Nested data cubes for olap. *LNCS 1552, Advances in Database Technologies*, 1998.

[4] Timothy Griffin and Leonid Libkin. Incremental maintenance of views with duplicates. *SIGMOD Conference*, pages 328–339, 1995.

[5] Ashish Gupta and Inderpal Mumick. *Materialized Views - Techniques, Implementation, and Applications.* The MIT Press, 1999.

[6] Ashish Gupta and Inderpal S. Mumick. Maintaining view incrementally. *SIGMOD Conference*, 1993.

[7] Ashish Gupta and Inderpal S. Mumick. Maintenance of materialized views: problems, techniques and applications. *IEEE Data Engineering Bulletin, special issues on materialized views and data warehousing*, 18(2), 1996.

[8] Richard Hull. A survey of theoretical research on typed complex database objects. In J. Paredaens, editor, *Databases*, pages 193–255. Academic Press, London, 1987.

[9] Informix. Informix-universal server administrator's guide, version 9.1. 1997. Informix Corporation.

[10] Mark Levene. *LNCS 595, The Nested Universal Relation Database Model.* Springer-verlag, Berlin, 1992.

[11] Jixue Liu and Millist Vincent. Containment and disjointedness in partitioned normal form relations. *Acta Informatica*, 38:325–342, 2002.

[12] Jixue Liu, Millist Vincent, and Mukesh Mohania. Implementation and performance analysis of incremental equations for nested relations. *IDEAS*, pages 398–404, 2000.

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

[14] M. A. Roth, H. F. Korth, and A. Silberschatz. Extended algebra and calculus for nested relational database. *ACM Transactions on Database Systems*, 13(4):389–417, 1988.

[15] M A Roth, H F Korth, and A Silberschatz. Null values in nested relational databases. *Acta Informatica*, 26(7):615–642, 1989.

[16] Mark A. Roth and James E. Kirkpatrick. Algebras for nested relations. *Data Engineering Bulletin*, 11(3):39–47, 1988.

[17] M. Stonebraker and Paul Brown. *Object-relational DBMSs tracking the next great wave.* Morgan Kaufmann Publishers, Inc. California, 1999.

[18] Michael Stonebraker and Dorothy Moore. *Object Relational DBMSs, the Next Great Wave.* Morgan Kaufman publishers Inc., 1996.

[19] M. Vincent, M. K. Mohania, and Y. Kambayashi. A self maintainable view maintenance technique for data warehouses. *COMAD*, pages 7–22, 1997.

[20] Jennify Widom. Research problems in data warehousing. *CIKM*, pages 25–30, 1995.

[21] Jun Yang and Jennifer Widom. Maintaining temporal views over non-historical information sources for data warehousing. *EDBT*, pages 389–403, 1998.

Received February, 2002