Efficiently Updating Data Warehouses and Data Marts

M. Tech. Project

By

Hoshi P. Mistry

Under the guidance of

Prof. Krithi Ramamritham

Prof. S. Sudarshan

Department of Computer Science and Engineering

Indian Institute of Technology

Bombay

October 1999

Motivation

- Materialized views → improves query response time.
- Primary assumption → the base relations remain unchanged (unrealistic).
- Several naive solutions have been provided.
  - Base relation updates are allowed only within an 'update window' (periodic).
  - Immediate view maintenance - does away with the periodicity of the update window.
- Need for a **clean** and a more **optimal** design for view maintenance, specifically for a data warehouse scenario.

Problem Specification

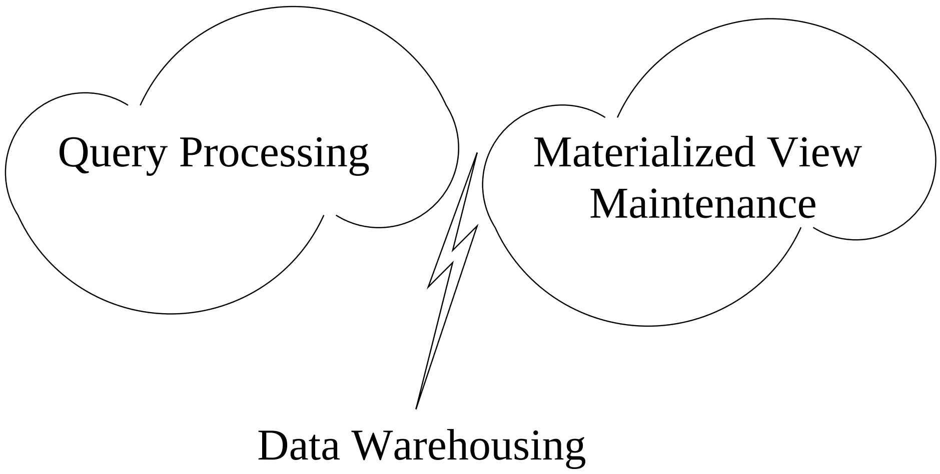


Figure 1: Query Processing and View Maintenance

- Problem Statement,
  Given that there exists a materialized view corresponding to a user specified query expression, the goal is to make the view uptodate with the logged updates in the most optimized fashion.
- Intuition $\rightarrow$ A query involving shared subexpressions can be optimized.

Fundamentals

- Deferred maintenance better for a data warehouse scenario.
- Deferred maintenence $\rightarrow$ View is upgraded after the updating transaction commits.
- Updates to the base relation $\rightarrow$ delta relations.
- Delta relations :
  - Same schema as the base relation.
  - An extra **timestamp** attribute that,
    * Facilitates delta relation maintenance.
    * Avoids propagation 'irrelevant' tuples.

Maintenance of Delta Relation

Propagation of Relevant Tuples

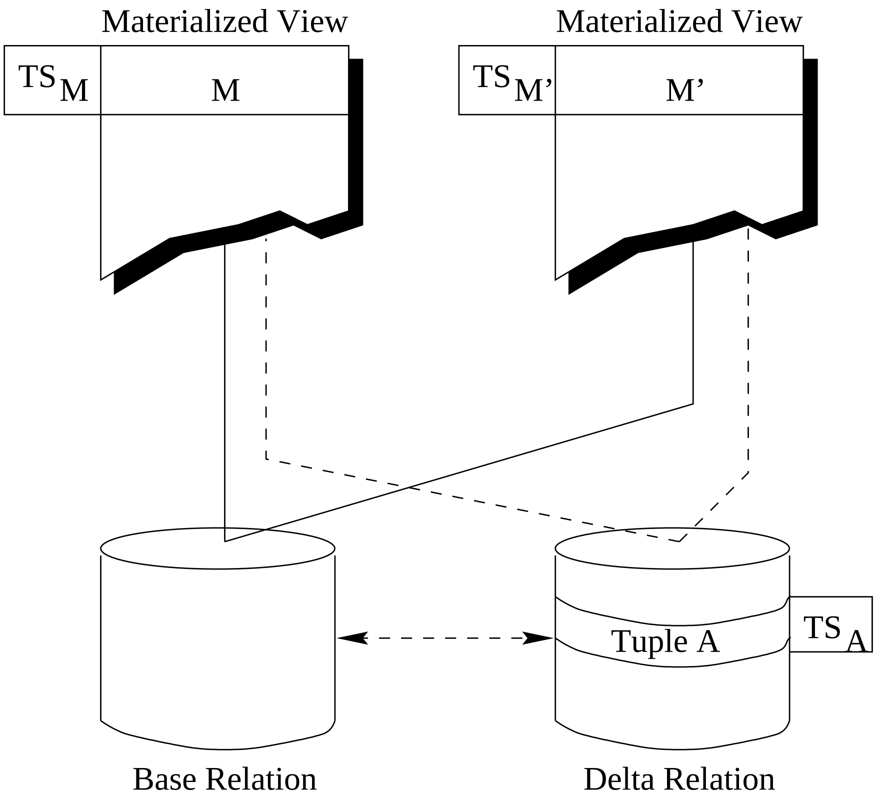


Figure 2: Views Sharing a Base Relation

- Delta Maintenance : $TS_A < \forall(TS_M, TS_{M'})$
- Propagation of Tuples : $TS_A > TS_M$

Incremental Update Propagation

- Compute only that part of the view that changes in response to the changes in the base relation. (heuristic)

- Example :
  Consider a view $V = A \bowtie B$, and base relation update logs : $\triangle A$ and $\triangle B$.

  Incremental update to the view :

  $$\triangle V = \triangle A \bowtie B \bigcup A \bowtie \triangle B \ \underset{-}{\bigcup} \ \triangle A \bowtie \triangle B$$

  Result :

  For Insert Operations : $V \bigcup \triangle V$
  For Delete Operations : $V - \triangle V$

Incremental Propagation (contd.)

- Ideally, both inserts and deletes occur.
- Updates can be treated as (insert + delete).
- Hence for base reln A : $\triangle A_i$, and $\triangle A_d$.
- Incremental expressions would be,

$$\triangle V_i = \triangle A_i \bowtie \mathrm{B} \bigcup \mathrm{A} \bowtie \triangle B_i \bigcup \triangle A_i \bowtie \triangle B_i$$
$$\triangle V_d = \triangle A_d \bowtie \mathrm{B} \bigcup \mathrm{A} \bowtie \triangle B_d - \triangle A_d \bowtie \triangle B_d$$
$$\triangle V_{Cross} = \triangle A_i \bowtie \triangle B_d \bigcup \triangle A_d \bowtie \triangle B_i$$

- Result expression,

$$\text{New View} = \mathrm{V} \bigcup \triangle V_i - \triangle V_d - \triangle V_{Cross}$$

- Tagging of update transactions,
  - Avoid working with two delta relns.
  - Modifications can be handled in one pass.

Incremental Propagation (contd.)

- Approach to avoid V_{Cross}.
  - Propagate all inserts first.

$$V = V \bigcup \triangle V_i$$

  - Update base relations with the insert delta(s). Takes care of V_{Cross}.

$$A = A \bigcup \triangle A_i$$
$$B = B \bigcup \triangle B_i$$

  - Now simply propagate V_d

$$V = V - \triangle V_d$$

  - Perform an extra operation

$$A = A - \triangle A_i$$
$$B = B - \triangle B_i$$

Update Propagation Approaches

- Two Approaches,
  - **Expression Generation Approach**
    Generates an incremental update differential expression which is subjected to the multi-query optimizer to find the extra plan cost of propagating the update changes from the base relations to the materialized view.
  - **Delta Propagation Approach**
    The second approach deals with the propagation of the relevant delta relation tuples, up the DAG, to the materialized views.

Multi-Query Optimizer

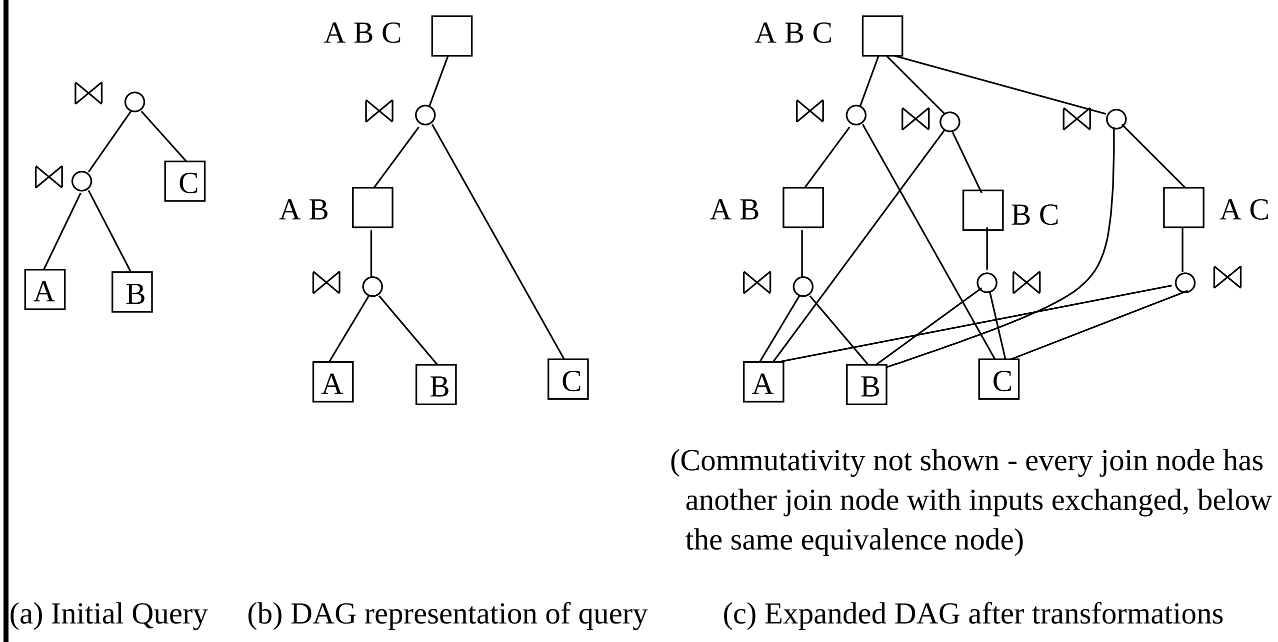


Figure 3: DAG : Multi-Query Optimizer

- Optimization done using MQO.
- Search for a materialized equivalence node corresponding the query expression.
- If not found → Re-computation.
- Else → perform incremental update (if appropriate).

Expression Generation Approach

- Consider a view $V = A \bowtie B \bowtie C$.
- Delta relations are : $\triangle A$, $\triangle B$, and $\triangle C$.
- Final result after update propagation,

$$(A \bigcup \triangle A) \bowtie (B \bigcup \triangle B) \bowtie (C \bigcup \triangle C)$$

$$\begin{aligned} \equiv\ & A \bowtie B \bowtie C \bigcup \\ & A \bowtie B \bowtie \triangle C \bigcup \\ & A \bowtie \triangle B \bowtie C \bigcup \\ & A \bowtie \triangle B \bowtie \triangle C \bigcup \\ & \triangle A \bowtie B \bowtie C \bigcup \\ & \triangle A \bowtie B \bowtie \triangle C \bigcup \\ & \triangle A \bowtie \triangle B \bowtie C \bigcup \\ & \triangle A \bowtie \triangle B \bowtie \triangle C. \end{aligned}$$

- After simplification we have,

$$\begin{aligned} \equiv\ & A \bowtie B \bowtie C \bigcup \\ & \triangle A \bowtie (B \bigcup \triangle B) \bowtie \underline{(C \bigcup \triangle C)} \bigcup \\ & A \bowtie \triangle B \bowtie \underline{(C \bigcup \triangle C)} \bigcup \\ & A \bowtie B \bowtie \triangle C \end{aligned}$$

Multiple Modification Types

- Intermittent occurrences of inserts and deletes are logged seperately

$$(A \bigcup \triangle A_i - \triangle A_d) \bowtie (B \bigcup \triangle B_i - \triangle B_d) \bowtie (C \bigcup \triangle C_i - \triangle C_d) \equiv$$

$$\begin{array}{l}
\equiv A \bowtie B \bowtie C \\
\bigcup \\
(\\
\quad \triangle A_i \bowtie (B \bigcup \triangle B_i) \bowtie (C \bigcup \triangle C_i) \bigcup \\
\quad A \bowtie \triangle B_i \bowtie (C \bigcup \triangle C_i) \qquad \bigcup \\
\quad A \bowtie B \bowtie \triangle C_i \\
) \\
- \\
(\\
\quad \triangle A_d \bowtie (B - \triangle B_d) \bowtie (C - \triangle C_d) \bigcup \\
\quad A \bowtie \triangle B_d \bowtie (C - \triangle C_d) \qquad \bigcup \\
\quad A \bowtie B \bowtie \triangle C_d \\
) \\
-V_{Cross}
\end{array}$$

Other Relational Operators

- Select and Project Operators

$$\sigma(\mathrm{A} \bigcup \triangle A_i \text{ - } \triangle A_d) = \sigma(\mathrm{A}) \bigcup \sigma(\triangle A_i) \text{ - } \sigma(\triangle A_d)$$

$$\prod(\mathrm{A} \bigcup \triangle A_i \text{ - } \triangle A_d) = \prod(\mathrm{A}) \bigcup \prod(\triangle A_i) \text{ - } \prod(\triangle A_d)$$

- For non-multiset projects → define counts.

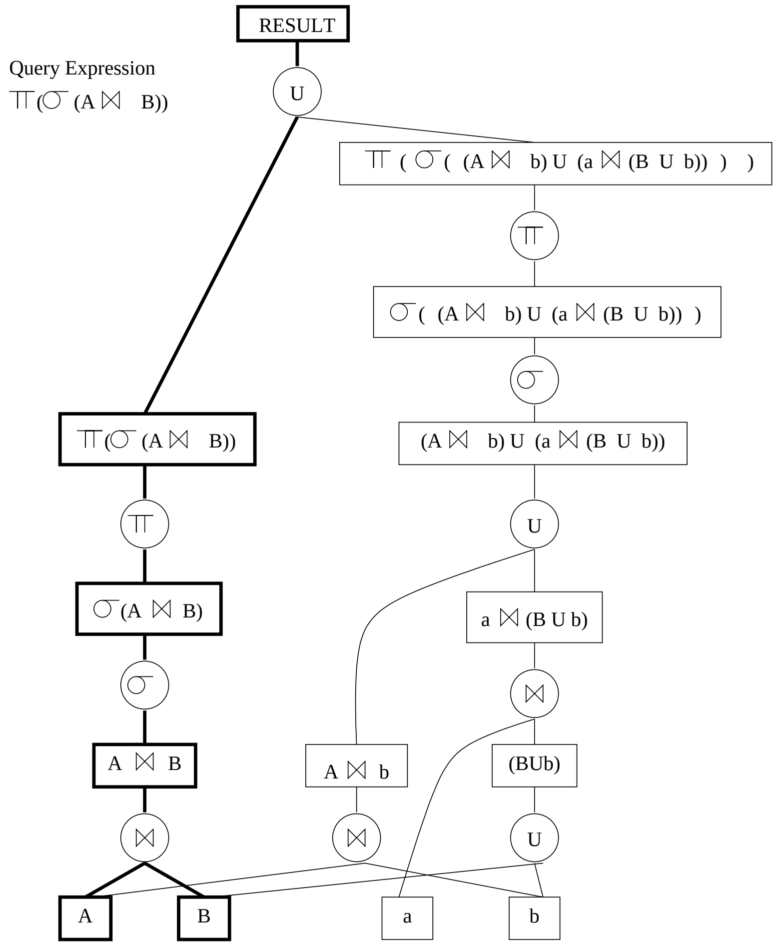


Figure 4: Expression tree for general query

Expression Generation Algorithm

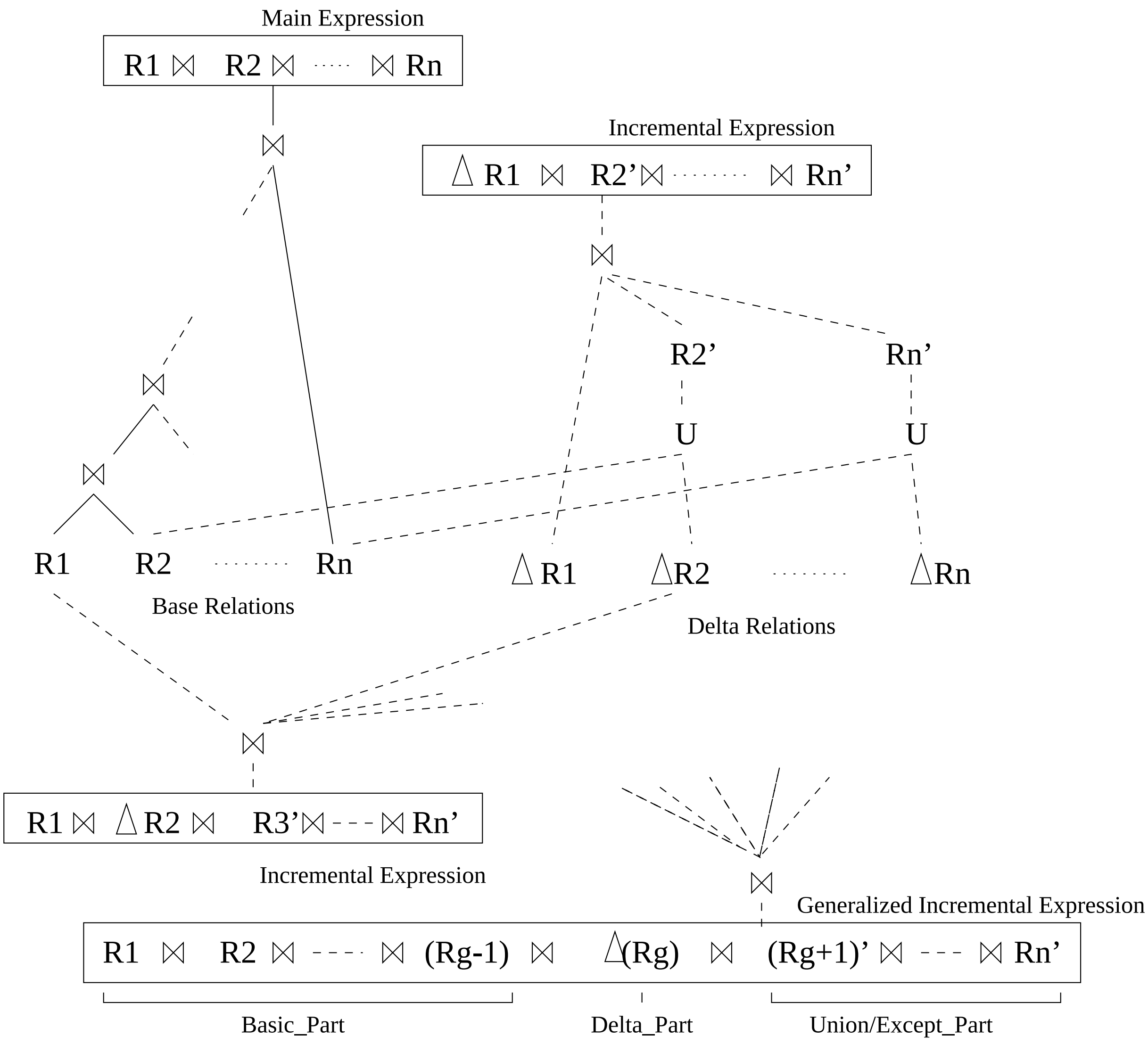


Figure 5: Generation of an Update Expression

Expression Generation - Aggregates

- Consider a materialized view V,

$$\text{V} = {}_{A.a1}G_{sum(B.a2)}(A \bowtie B)$$

- Requirement of a *Group Merge* operator 'M' to merge the aggregations from the delta relation tuples with the aggregations from the base relation tuples.

- Incremental Update Expression,

$$\begin{aligned} \text{V'} = {}& {}_{A.a1}G_{sum(B.a2)}(A \bowtie B) \quad \text{M} \\ & {}_{A.a1}G_{sum(B.a2)}((\triangle A_i \bowtie (B \bigcup \triangle B_i)) \bigcup (A \bowtie \triangle B_i)) \quad \text{M} \\ & {}_{A.a1}G_{sum(B.a2)}((\triangle A_d \bowtie (B - \triangle B_d)) \bigcup (A \bowtie \triangle B_d)) \end{aligned}$$

A Different Approach

- Given a query Q.
- Come up with Q_x, Q_y, and Q_z.
- If X and Z are updated → apply Q_x and Q_z

$$
\begin{aligned}
Q \quad = \quad & SELECT \quad X.a, Y.b, Z.c \\
& FROM \quad X, Y, Z \\
& WHERE \quad X.a = Y.a \quad AND \quad Y.b = Z.b
\end{aligned}
$$

$$
\begin{aligned}
Q_x \quad = \quad & SELECT \quad \triangle\, X.a, Y.b, Z.c \\
& FROM \quad \triangle\, X, Y, Z \\
& WHERE \quad \triangle\, X.a = Y.a \quad AND \quad Y.b = Z.b
\end{aligned}
$$

$$
\begin{aligned}
Q_y \quad = \quad & SELECT \quad X.a, \triangle Y.b, Z.c \\
& FROM \quad X, \triangle Y, Z \\
& WHERE \quad X.a = \triangle Y.a \quad AND \quad \triangle\, Y.b = Z.b
\end{aligned}
$$

$$
\begin{aligned}
Q_z \quad = \quad & SELECT \quad X.a, Y.b, \triangle Z.c \\
& FROM \quad X, Y, \triangle Z \\
& WHERE \quad X.a = Y.a \quad AND \quad Y.b = \triangle Z.b
\end{aligned}
$$

Delta Propagation Approach

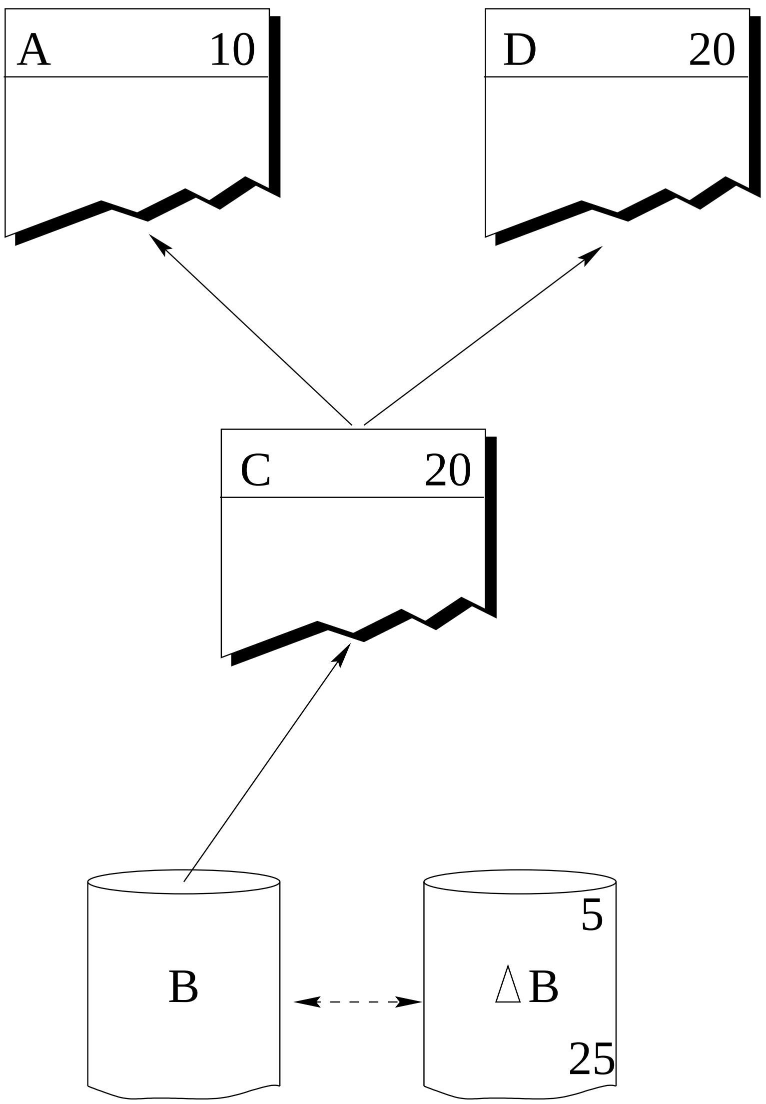


Figure 6: Number of tuples to be propagated

Delta Propagation - In a nutshell

- $Q = R1 \bowtie R2 \bowtie R3$. ($\triangle R1$, $\triangle R2$, $\triangle R3$)
- DAG required $\rightarrow$ Expand DAG (no updates).
- Algorithm propagates a delta at a time.
- If Expression DAG is like fig A,

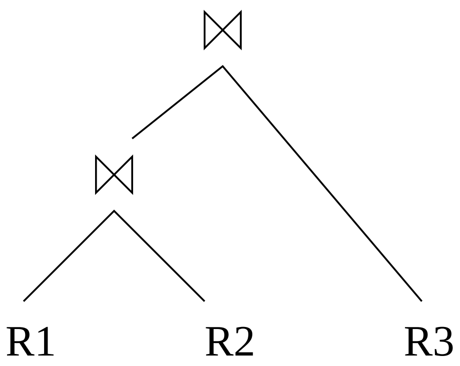


Fig A

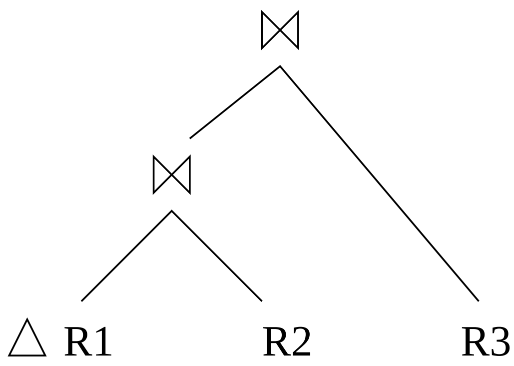


Fig B

- $\triangle R1$ can be propagated as, (fig B).

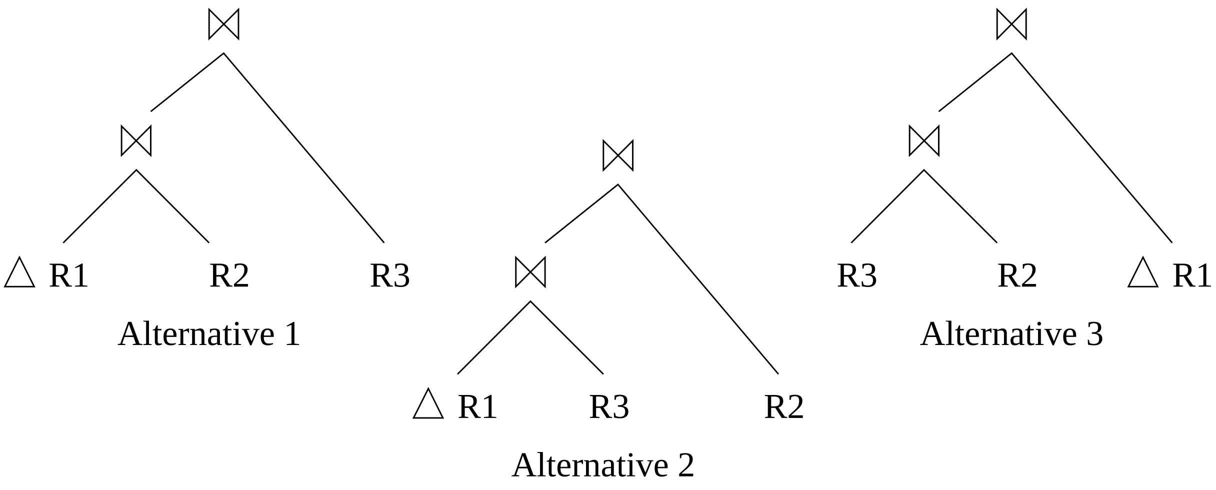
$$\triangle R1 \bowtie R2 \bowtie R3$$

- Similarly we have,

$$R1 \bowtie \triangle R2 \bowtie R3$$
$$R1 \bowtie R2 \bowtie \triangle R3$$

Various Combinations

- Consider $\triangle R1 \bowtie R2 \bowtie R3$.
- Various combinations are,



- Apply MQO, and find cheapest.

Related Issues

- Relative orders between $\triangle$Ri and $\triangle$Rj missed.
- Order in which $\triangle$Ri(s) are considered is important.
- Cardinality of $\triangle$Ri is assumed to be small.
- Use of shared sub-expressions is not seen.
- Further analysis giving yet another heuristic that caters to,
  - Large cardinality of $\triangle$Ri.
  - Sharing of sub-expressions.

Analytical Comparison

- Good comparison metric,

total time = find_time(plan) + execute_time(plan)

Future Work

- Both Expression generation and Delta Propagation approaches have been partly implemented.
- Some enhancement ideas which need further investigation and thought.

  1. Materialization of intermediate nodes.
  2. Locality of the updates with respect to the base relations.
  3. Propagation of updates in the reverse direction (from Mat views to the base relations).
  4. Investigation of literature for solutions to self-maitainable views, and their compatibility with our architecture.

System Scenario

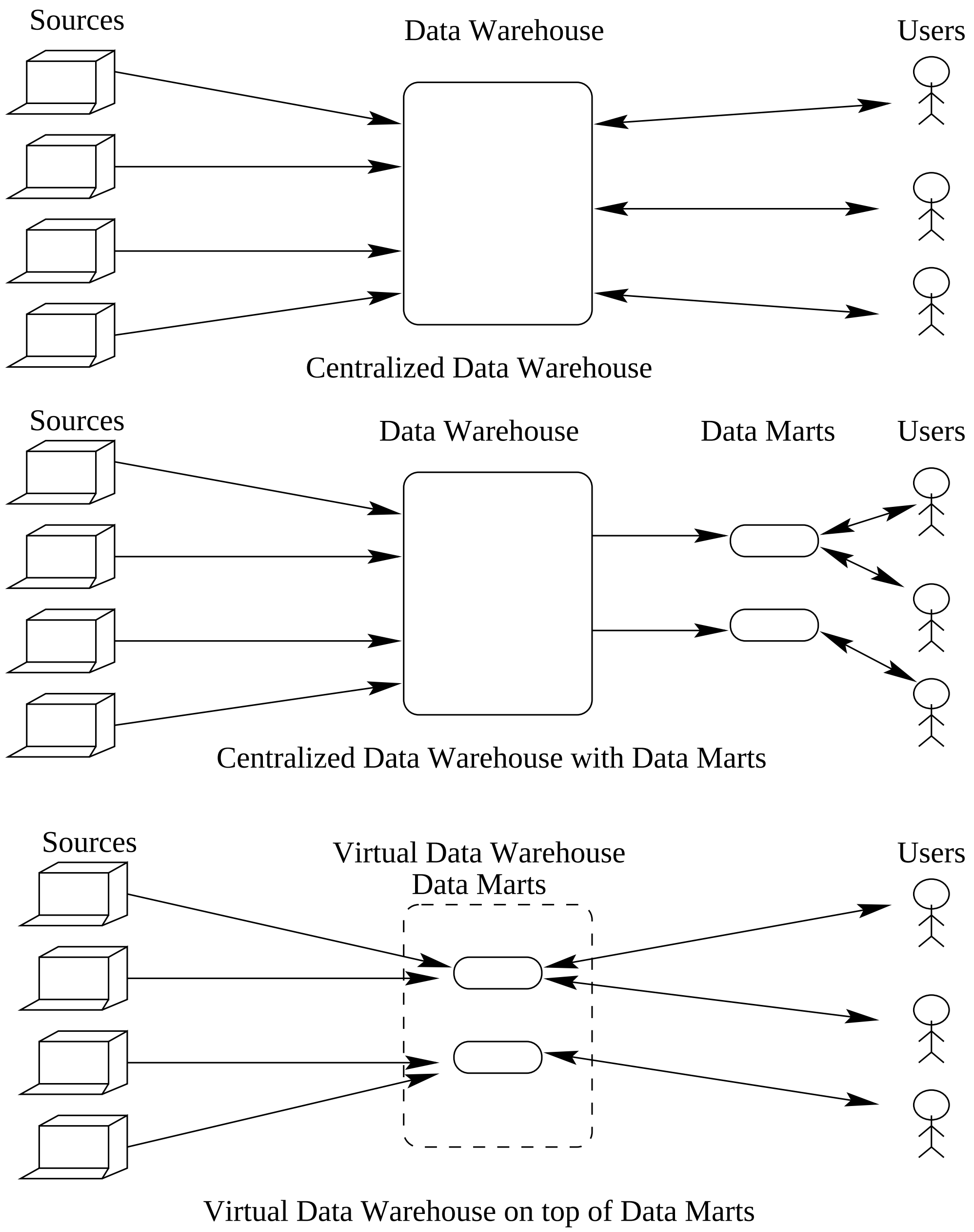


Figure 7: Data Warehouse Architectures

Expr Generation Approach (contd.)

- Similarly for the delete operation,

$$
\begin{aligned}
&(A - \triangle A) \bowtie (B - \triangle B) \bowtie (C - \triangle C) \\
&\equiv A \bowtie B \bowtie C \quad - \\
&\quad (\\
&\quad\quad A \bowtie B \bowtie \triangle C \quad \bigcup \\
&\quad\quad A \bowtie \triangle B \bowtie C \quad - \\
&\quad\quad A \bowtie \triangle B \bowtie \triangle C \quad \bigcup \\
&\quad\quad \triangle A \bowtie B \bowtie C \quad - \\
&\quad\quad \triangle A \bowtie B \bowtie \triangle C \quad - \\
&\quad\quad \triangle A \bowtie \triangle B \bowtie C \quad \bigcup \\
&\quad\quad \triangle A \bowtie \triangle B \bowtie \triangle C \\
&\quad) \\
&\equiv A \bowtie B \bowtie C \quad - \\
&\quad (\\
&\quad\quad \triangle A \bowtie (B - \triangle B) \bowtie \underline{(C - \triangle C)} \quad \bigcup \\
&\quad\quad A \bowtie \triangle B \bowtie \underline{(C - \triangle C)} \quad \bigcup \\
&\quad\quad A \bowtie B \bowtie \triangle C \\
&\quad)
\end{aligned}
$$

Expression Generation (contd)

The incremental expression is composed of several units. Each unit consists of three parts, viz.

1. Left deep join of the base relations. (0 or many base relations connected via a join operation).
2. Exactly one delta relation.
3. Base relation Union/Except the corresponding delta relations. (0 or many instances connected via a join operation).

Expression Generation (contd)

1. Let there be a query involving a join of 'n' base relations R_1, R_2, $\cdots$, R_n.
2. In the worst case assume that all the 'n' relations are updated before the query is fired. Thus $\exists$ $\triangle R_{i_1}$, $\triangle R_{i_2}$, $\cdots$, $\triangle R_{i_n}$, and $\triangle R_{d_1}$, $\triangle R_{d_2}$, $\cdots$, $\triangle R_{d_n}$
3. LET main_expr = $R_1 \bowtie R_2 \bowtie \cdots \bowtie R_n$.
4. LET incr_expr_insert = $\emptyset$.
5. LET incr_expr_delete = $\emptyset$.
6. LET basic_part = $\emptyset$.
7. FOR relation pointer rptr = 1 to n do
   (a) IF rptr > 1 THEN basic_part = basic_part $\bowtie R_{rptr-1}$
   (b) delta_part_insert = $\triangle R_{i_{rptr}}$.

(c) delta_part_delete = $\triangle R_{d_{rptr}}$.

(d) union_part = $(R_{rptr+1} \bigcup \triangle R_{i_{rptr+1}}) \bowtie \cdots \bowtie (R_n \bigcup \triangle R_{i_n})$.

(e) except_part = $(R_{rptr+1}$ - $\triangle R_{d_{rptr+1}}) \bowtie \cdots \bowtie (R_n$ - $\triangle R_{d_n})$.

(f) incr_expr_insert = incr_expr_insert $\bigcup$ (basic_part $\bowtie$ delta_part_insert $\bowtie$ union_part).

(g) incr_expr_delete = incr_expr_delete $\bigcup$ (basic_part $\bowtie$ delta_part_delete $\bowtie$ except_part).

ENDFOR.

8. result = main_expr $\bigcup$ incr_expr_insert - incr_expr_delete.

Expression Generation (Aggregates)

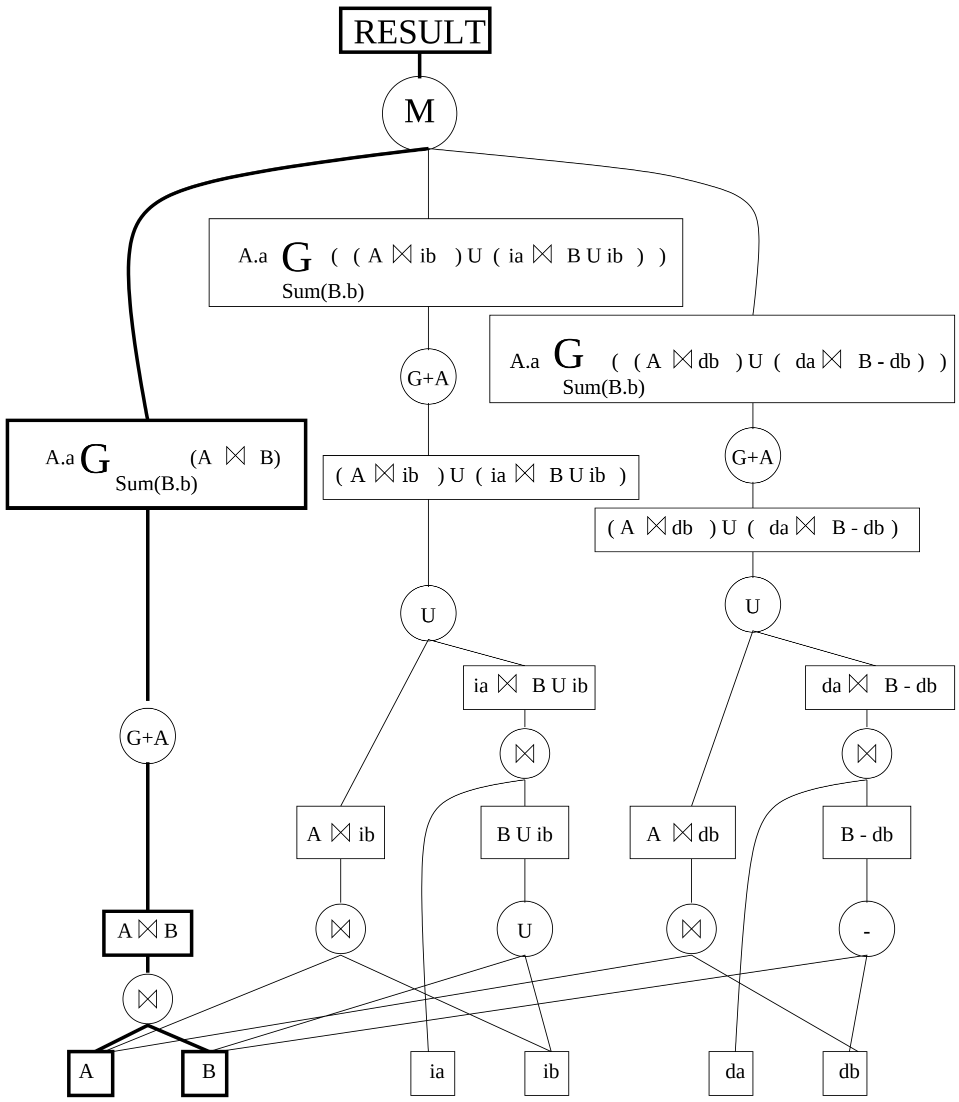


Figure 8: Expression Generation (Aggregates)

Group Merge Algorithm

1. LET final_result = ∅.
2. LET V = original contents of the mat view
3. LET V_in = Incremental update contents.
4. LET V_in = ∅.
5. FOR each distinct group-by attribute value 'g' in V_in do
   (a) Search for the same value in a group-by attribute in V.
   (b) IF not found THEN
       IF action == insert THEN
           insert the tuple containing g in V.
       ELSE // action == delete
           flag an error message.
       ENDIF
     ELSE // value found
     SWITCH(aggregate_type)

Count

i. CASE(count) :

A. IF action == insert THEN
    V(count) += V_in(count).
    ELSE // action == delete
    V(count) -= V_in(count).
    ENDIF

B. IF count == 0 THEN delete tuple from V.

C. IF count < 0 THEN flag error.

Sum

ii. CASE(sum) :
    IF action == insert THEN
    V(sum) += V_in(sum).
    ELSE // action == delete
    V(sum) -= V_in(sum).
    ENDIF

Min/Max

iii. CASE(min/max) :

A. IF action == delete THEN

- IF V_in(mn/mx) == V(mn/mx)
  - Find the next min/max from base relation and replace in V.
  - IF no such value can be found THEN delete the tuple from V.

  ELSE // Value in V_in is > / < that in V.

  No action.

  ENDIF

ENDIF

B. IF action == insert THEN

IF V_in(mn/mx) </> V(mn/mx)

V(min/max) = V_in(min/max).

ELSE

No action.

ENDIF

ENDIF

Average

iv. CASE(average) :
- Both V and V_in maintain the sum and the count aggregates in addition to maintaining the average.
- Merge action is taken on the sum and the count as discussed in steps i and ii above.
- The average is recomputed by dividing the sum by the count.
- Replace the old average with the new average.

ENDIF

ENDFOR

6. final_result = V

Implementation Results

- MQO improves the quality of plan.
- Average improvement is > 75%.
- Worst case improvement > 71%.

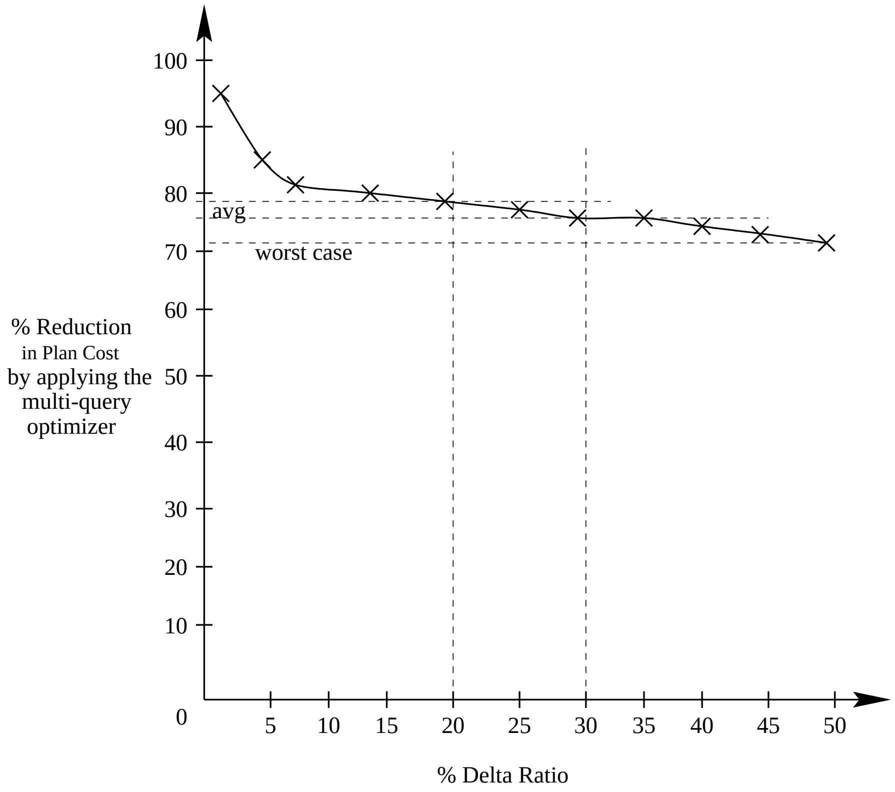


Figure 9: Delta Ratio v/s Plan Cost Reduction(MQO)

Delta Relation Timestamps

- Delta Relations have timestamps.
- Tuple (delta $\rightarrow$ base) iff
  timestamp of tuple $<$ timestamp of each derived materialized view.
- Functions of the timestamp attribute,
  - Identifies the time instant the tuple was created.
  - Avoids the propagation of irrelevant tuples.
  - Reduces the number of tuples to be propagated.

Illustrative Example

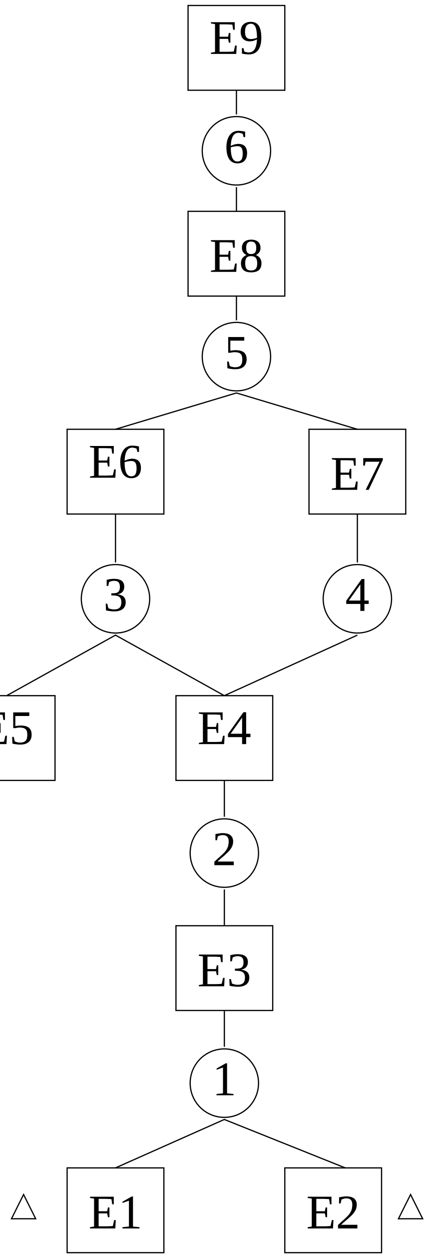


Figure 10: Different Execution Steps

Approach for Large Delta

- V = R1 ⋈ R2 ⋈ R3 (△R1, △R2, △R3).
- Suppose all △s could fit in memory.

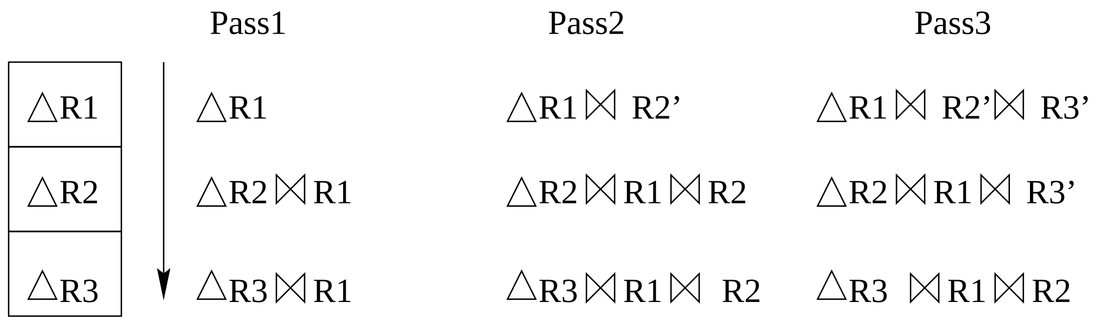


Figure 11: Algorithm Passes

1. Read R1.
   Perform Pass 1.
2. Read R2.
   Compute R2' = R2 $\bigcup$ △R2
   Perform Pass 2.
3. Read R3.
   Compute R3' = R3 $\bigcup$ △R3
   Perform Pass 3.

Approach for Large Delta

- Order of choosing Ri is crucial.

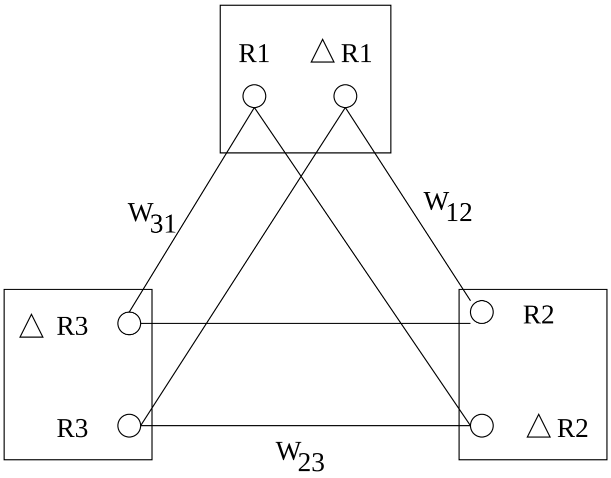


Figure 12: Join Graph

- Choose Ri such that

$$\sum_{\forall i, i \neq j} W_{ij} = \min\left(\sum_{\forall p, p \neq j} W_{pj} \right)$$

where W_{ij} and W_{pj} are weights from R_j to $\triangle R_i$ and $\triangle R_p$ respectively.

- After each pass, recompute the graph & weights.

Sample Incremental Query

- Basic Query Expression

```
SELECT PART.P_PARTKEY
FROM PART, SUPPLIER, PARTSUPP
WHERE PART.P_PARTKEY = PARTSUPP.PS_PARTKEY
AND SUPPLIER.S_SUPPKEY = PARTSUPP.PS_SUPPKEY ;
```

- Incremental Query Expression

```
SELECT dPART.P_PARTKEY
FROM dPART, ( SELECT SUPPLIER.S_SUPPKEY FROM
SUPPLIER
UNION SELECT dSUPPLIER.S_SUPPKEY FROM dSUPPLIER )
AS S ,
( SELECT PARTSUPP.PS_PARTKEY, PARTSUPP.PS_SUPPKEY
FROM PARTSUPP
UNION SELECT dPARTSUPP.PS_PARTKEY,
dPARTSUPP.PS_SUPPKEY FROM dPARTSUPP ) AS P
WHERE dPART.P_PARTKEY = P.PS_PARTKEY
AND S.S_SUPPKEY = P.PS_SUPPKEY
UNION
SELECT PART.P_PARTKEY
FROM PART , dSUPPLIER, ( SELECT
PARTSUPP.PS_PARTKEY, PARTSUPP.PS_SUPPKEY
FROM PARTSUPP
UNION SELECT dPARTSUPP.PS_PARTKEY,
dPARTSUPP.PS_SUPPKEY FROM dPARTSUPP ) AS P
WHERE PART.P_PARTKEY = P.PS_PARTKEY
AND dSUPPLIER.S_SUPPKEY = P.PS_SUPPKEY
UNION
SELECT PART.P_PARTKEY
FROM PART, SUPPLIER, dPARTSUPP
WHERE PART.P_PARTKEY = dPARTSUPP.PS_PARTKEY
AND SUPPLIER.S_SUPPKEY = dPARTSUPP.PS_SUPPKEY
;
```

Expression for V_{Cross}

$$
\begin{aligned}
-V_{Cross} = - \{ & \\
& (\triangle A_i \bowtie (\underline{\mathrm{B} \bigcup \triangle B_i}) \bowtie \triangle C_d) \\
& \bigcup (\triangle A_i \bowtie \triangle B_d \bowtie (\underline{\mathrm{C} \bigcup \triangle C_i}) \\
& \bigcup (\triangle A_d \bowtie \triangle B_i \bowtie (\underline{\mathrm{C} \bigcup \triangle C_i}) \\
& \bigcup (\triangle A_d \bowtie (\underline{\mathrm{B} \bigcup \triangle B_i}) \bowtie \triangle C_i) \\
& \bigcup (\mathrm{A} \bowtie \triangle B_i \bowtie \triangle C_d) \\
& \bigcup (\mathrm{A} \bowtie \mathrm{B} \bowtie \triangle C_i) \\
& - (\triangle A_i \bowtie \triangle B_d \bowtie \triangle C_d) \\
& - (\triangle A_d \bowtie \triangle B_i \bowtie \triangle C_d) \\
& \}
\end{aligned}
$$

Incremental Update - Outer Join

- Consider a view

```
CREATE VIEW V AS
        SELECT X
        FROM R(Y) ||&|| S(Z)
        WHERE g(Y, Z);
```

- The △V can be computed by,

```
SELECT X                       SELECT X
FROM △R(Y) ||& S(Z)            FROM R^v(Y) &|| △S(Z)
WHERE g(Y, Z);                 WHERE g(Y, Z);
      (a)                            (b)
```

- Summary of executions of (a) and (b)

Delta Query	Effect on View V	Side Effect
query (a)	insert r^{+}.s	delete NULL.s
△R ||& S	insert r^{+}.NULL	
	delete r^{-}.s	insert NULL.s
	delete r^{-}.NULL	
query (b)	insert $r^{v}.s^{+}$	delete r.NULL
R^{v}(Y) &|| △S	insert NULL.s^{+}	
	delete $r^{v}.s^{-}$	delete r.NULL
	delete NULL.s^{-}	

Self Maintainable Views

- Views that can be maintained using only the contents of the view, and delta relations, without accessing the base relations are called Self Maintainable views.
- Insertions
  - All SP views are self-m wrt insertions.
  - SPJ view using self joins are self-m if every join is based on key attrs.
- Deletes
  - If a key of all the base relns is a projected attr, then the view is self-m wrt deletes.
- Updates
  - View is self-m wrt updates to non-exposed attrs iff view is self-m wrt to deletes.

Outer Join Query : View R ||& S, is self-m wrt all types of modifications to S iff,

1. Keys of R and S are projected.
2. All exposed attrs of R are projected.

Complexity Analysis

- Complexity Analysis is done on the basis of,
  - # of distinct combinations (orders).
  - # of operators.
- # of distinct combinations.
  - Consider Q = R1 $\bowtie$ R2 $\bowtie$ R3 $\rightarrow$ R1R2R3.
  - Consider the Associative order (R1 R2) R3.
  - Applying Commutative transformations,
    (R2 R1) R3, R3 (R1 R2), R3 (R2 R1).
  - Similarly for other Associative orders we have,
    (R1 R3) R2, (R3 R1) R2, R2 (R1 R3), R2 (R3 R1).
    and
    (R2 R3) R1, (R3 R2) R1, R1 (R2 R3), R1 (R3 R2).
  - Hence for 3 relations the # of distinct join orders are 12.
  - Hence the # of distinct join orders are of order $\mathbf{O(n*2^{n-1})}$.
- # of distinct operators.
  - Each of the 12 distinct join orders have 2 '$\bowtie$' operators.
  - Hence the # of distinct operators are 2 * 12 = 24.
  - Hence the # of distinct operators are of order $\mathbf{O(3^n)}$.

Multiple Modification Types (contd)

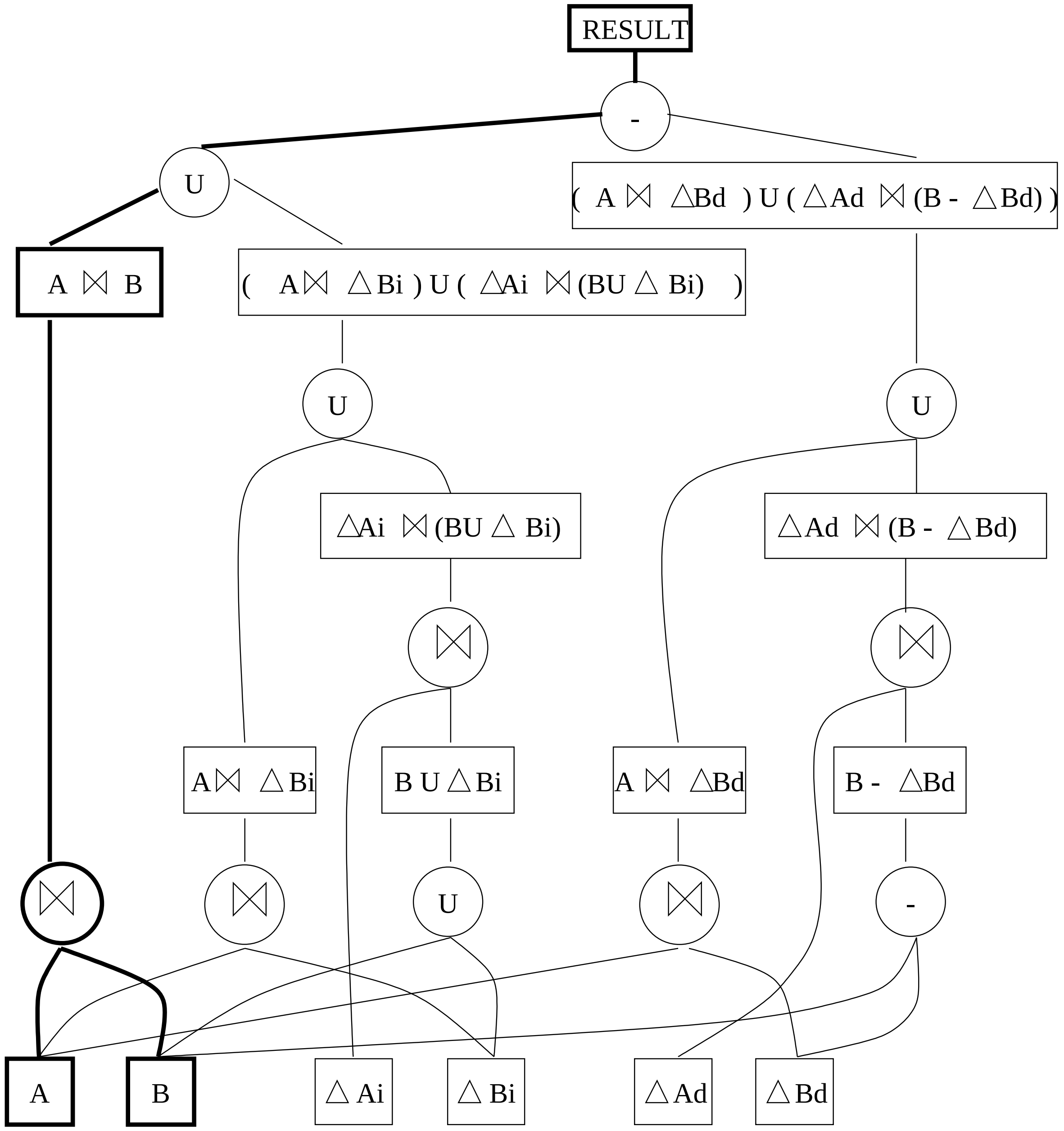


Figure 13: Expression tree for Basic and Incremental query expressions

Optimized Update Heuristic

1. Optimize the expression using relational algebra transformations.
2. Generate the incremental update expression.
3. Apply the multi-query optimizer to obtain the best plan.

Relational Algebra Transformations

1. Perform the (σ) operation as early as possible.
2. Perform ($\prod$) operation early.
3. Determine which (σ) and ($\bowtie$) operation will produce the smallest results and perform them first.
4. Replace $\sigma(\times)$, by a $\bowtie$ operation.

Implementation Issues

Major components :

1. An *Expression Search* routine $\rightarrow$ search for the marked (materialized) equivalence node
2. A *parser*
   (a) Parse the user specified query.
   (b) Generate the incremental expression (string manipulations).
3. An *optimizer* $\rightarrow$ generate the most optimal plan

Incremental Query in SQL

Consider the expression, (Assume only inserts)

$$(\mathrm{A} \bigcup \triangle A_i) \bowtie (\mathrm{B} \bigcup \triangle B_i) \bowtie (\mathrm{C} \bigcup \triangle C_i) \equiv$$

$$\begin{aligned}
\equiv\ & \mathrm{A} \bowtie \mathrm{B} \bowtie \mathrm{C} \\
& \bigcup \\
& (\\
& \quad \triangle A_i \bowtie (\mathrm{B} \bigcup \triangle B_i) \bowtie (\mathrm{C} \bigcup \triangle C_i) \bigcup \\
& \quad \mathrm{A} \bowtie \triangle B_i \bowtie (\mathrm{C} \bigcup \triangle C_i) \qquad \bigcup \\
& \quad \mathrm{A} \bowtie \mathrm{B} \bowtie \triangle C_i \\
&)
\end{aligned}$$

Important Features :

1. Query nesting in the FROM clause.
2. Rename feature of SQL using the AS construct.
3. The UNION construct (EXCEPT in case of delete).
4. Inclusion of delta relations.

Incremental Query in SQL (contd)

The SQL equivalent would be,

SELECT * FROM A, B, C WHERE A.a = B.a AND B.b = C.b

UNION

SELECT * FROM $\triangle A_i$,

(SELECT * FROM B UNION SELECT * FROM $\triangle B_i$) AS B',

(SELECT * FROM C UNION SELECT * FROM $\triangle C_i$) AS C'

WHERE $\triangle A_i$.a = B'.a AND B'.b = C'.b

UNION

SELECT *

FROM A, $\triangle B_i$,

(SELECT * FROM C UNION SELECT * FROM $\triangle C_i$) AS C'

WHERE A.a = $\triangle B_i$.a AND $\triangle B_i$.b = C'.b

UNION

SELECT *

FROM A, B, $\triangle C_i$

WHERE A.a = B.a AND B.b = $\triangle C_i$.b

Implementation Results

- Average Speedup is roughly 4.

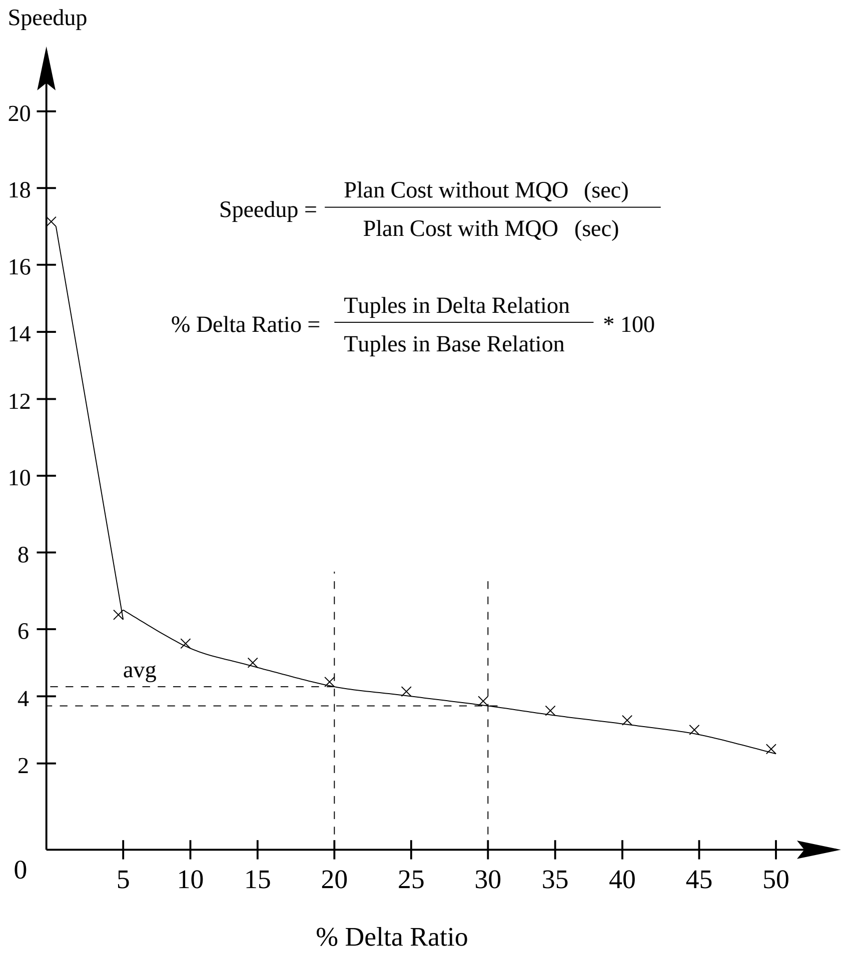


Figure 14: Delta Ratio v/s Speedup(MQO)

- SELECT PART.P_PARTKEY
  FROM PART, SUPPLIER, PARTSUPP
  WHERE PART.P_PARTKEY = PARTSUPP.PS_PARTKEY
  AND SUPPLIER.S_SUPPKEY = PARTSUPP.PS_SUPPKEY ;

Implementation Results (contd)

The following compares,

- The recomputation cost,
- The incremental Plan Cost.(MQO & without)
- MQO performs well on the shared expressions.

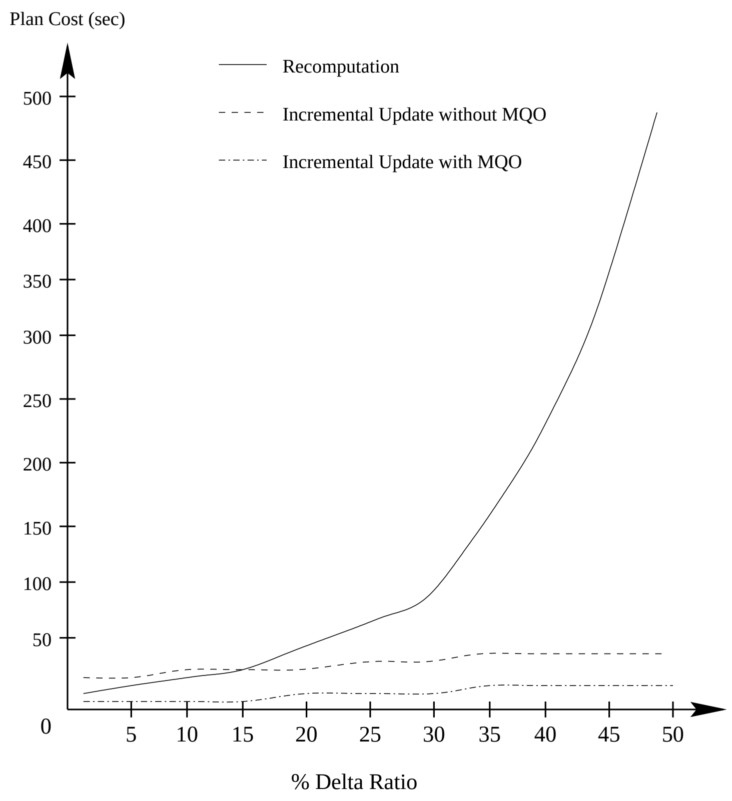


Figure 15: Performance Measurements

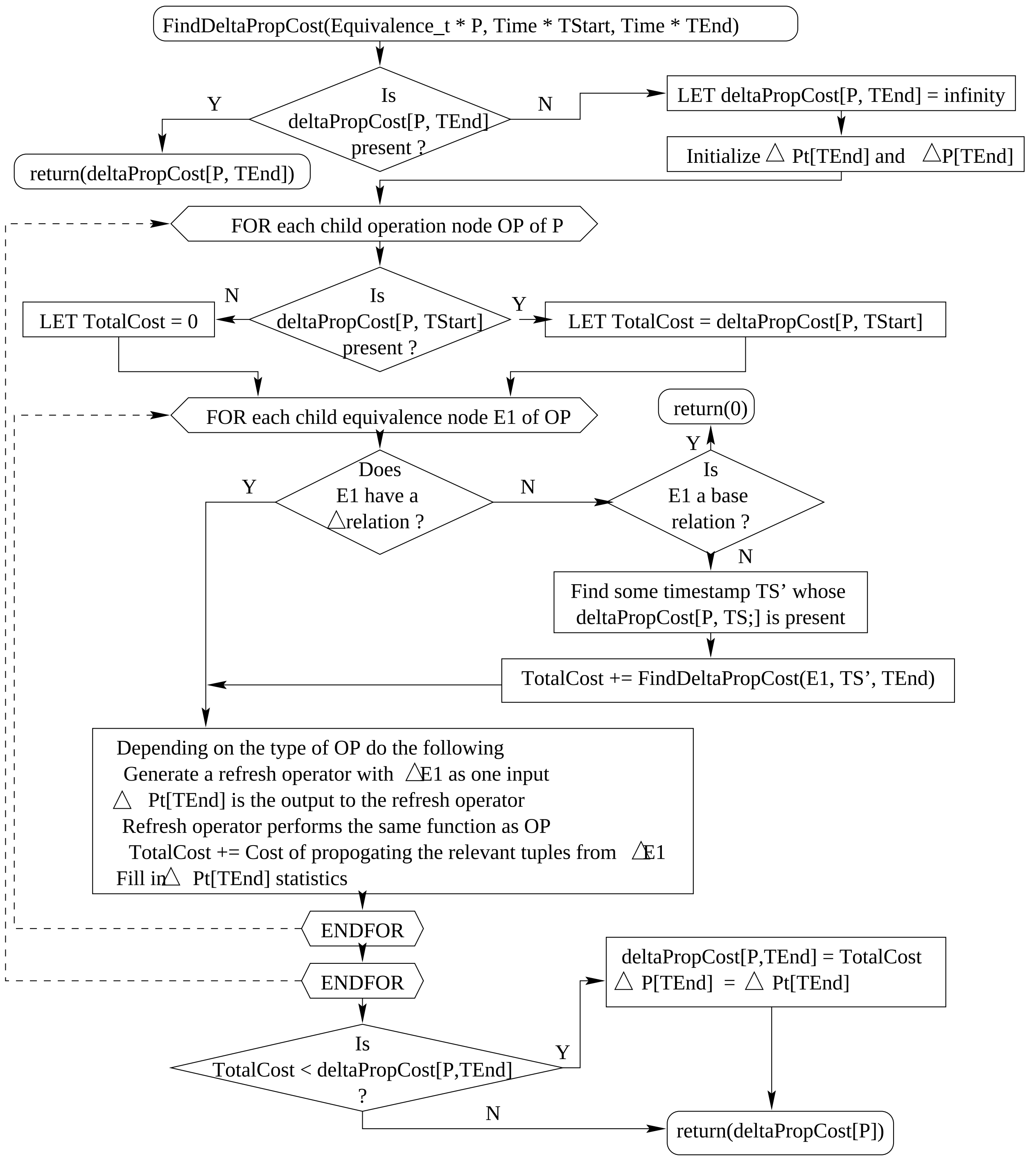
Query Execution

Suppose we have relations X, Y, and Z. Now suppose that relations X and Z (only) undergo updates.

We can find the incremental update to the materialized view by,

- $Q_x \longrightarrow$ incr_part1.
- UPDATE X WITH (SELECT * FROM $\triangle$X). This takes care of ($\triangle X \bowtie \triangle Z$) in the next step.
- $Q_z \longrightarrow$ incr_part2.
- incr_part1 UNION incr_part2.
- X - $\triangle$X.

Delta Propagation - Algorithm



Extension

- Apply MQO $\rightarrow$ generate Best Plan.
- Order R1, R2, ..., Rn according to Plan.
- Remove Ri whose $\triangle$Ri is to be propagated.
- Find Rj s.t, $\triangle$Ri $\bowtie$ Rj gives best result.
- Perform Rj' = $\triangle$Ri $\bowtie$ Rj
- Treat Rj' as Rj in the best plan.
- Perform the join giving smallest result, first.