Incremental View Maintenance

CPS 296.1

Topics in Database Systems

Virtual views

- A view is defined by a query over base tables
  - Example: CREATE VIEW *V* AS SELECT … FROM *R*, *S* WHERE …;
- A view can be queried just as a normal table
  - Example: SELECT * FROM *V*;
- Traditionally, database views are virtual
  - DBMS stores the view definition query instead of contents
  - Queries that reference views are rewritten (“expanded”) using the view definition queries to reference base tables directly
- Why use virtual views?
  - Access control
  - Hiding complexity
  - Logical data independence

Materialized views

- A view can be materialized, i.e., its contents can be pre-computed and stored by the DBMS
- Why materialized views?
  - Query performance
  - Reliability (if materialized elsewhere)
- Issues
  - View maintenance: how to maintain the consistency between base data and materialized results
  - View selection: how to choose what views to materialize
  - Answering query using views: how to rewrite queries to make use of the materialized results

View maintenance

- When base data changes, materialized views need to maintained
  - Re-computation
  - Incremental maintenance: compute and apply only the incremental changes to the materialized views
- Techniques are widely applicable
  - Derived data maintenance (warehouse, cache, etc.)
  - Integrity constraint checking

➢ A theoretical introduction: Griffin and Libkin. “Incremental Maintenance of Views with Duplicates.” *SIGMOD*, 1995

- Many practical issues to be addressed next week

Review of bag algebra

- Closer to SQL than relational algebra
- A table is a bag (or multiset)
  - Duplicate tuples are allowed
  - The number of duplicates matters
- Bag algebra operators
  - σ_p (selection), π_A (projection), × (cross product)
    - Above three are the most commonly used
  - $\oplus$ (additive union), $\ominus$ (monus), min (minimum intersection), max (maximum union)
  - ϵ (duplicate elimination)

Bag algebra operators (slide 1)

- Selection: $\sigma_p\ (S)$
  - Filters out tuples
  - Preserves duplicates (those that pass *p*)
- Projection: $\pi_A\ (S)$
  - Projects away attributes not in *A*
  - Preserves duplicates; that is, $|\ S\ | = |\ \pi_A\ (S)\ |$
- Cross product: $R \times S$
  - Pairs up tuples
  - count$((r, s), R \times S)$ = count$(r, R) \cdot$ count(s, S)

Bag algebra operators (slide 2)

- Additive union: $R \oplus S$
  - count(x, $R \oplus S$) = count(x, R) + count(x, S)
  - Example: {2 apples} $\oplus$ {3 apples} = {5 apples}
- Monus: $R \ominus S$
  - count(x, $R \ominus S$) = count(x, R) – count(x, S); or 0 if count(x, R) < count(x, S)
  - Example: {2 apples, 2 bananas} $\ominus$ {3 apples, 1 banana} = {1 banana}
- Duplicate elimination: $\in(S)$
  - count(x, $\in(S)$) = 1; or 0 if x is not in S at all
  - Example: $\in$({2 apples, 2 bananas}) = {1 apple, 1 banana}

Bag algebra operators (slide 3)

- Minimum intersection: R min S
  - count(x, R min S) = min(count(x, R), count(x, S))
  - Example: {2 apples} min {3 apples} = {2 apples}
  - Can you define it using the other operators?
    - R min $S = R \ominus (R \ominus S)$
- Maximum union: R max S
  - count(x, R max S) = max(count(x, R), count(x, S))
  - Example: {2 apples} max {3 apples} = {3 apples}
  - Can you define it using the other operators?
    - R max $S = R \oplus (S \ominus R) = (R \oplus S) \ominus (R$ min $S)$

Describing changes to base tables

- A transaction t modifies $R_1, \ldots, R_n$ in one atomic step

  $R_1 \leftarrow (R_1 \ominus \nabla R_1) \oplus \Delta R_1$

  … …

  $R_n \leftarrow (R_n \ominus \nabla R_n) \oplus \Delta R_n$
- ∇R_i contains
  - Tuples deleted by t from R_i
  - Old contents of the R_i tuples updated by t
- ΔR_i contains
  - Tuples inserted by t into R_i
  - New contents of the R_i tuples updated by t

Pre-expression

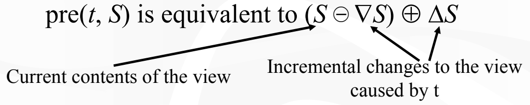
- $S(R_1, \ldots, R_n)$ is a bag algebra query expression defining a view
- Pre-expression of S w.r.t. t, pre(t, S), is defined as

  $S\,(\,(R_1 \ominus \nabla R_1) \oplus \Delta R_1, \ldots, (R_n \ominus \nabla R_n) \oplus \Delta R_n\,)$
  - Intuitively represents full re-computation of the view
  - Uses the current state of the database before the transaction is applied
  - Computes the would-be contents of the view after the transaction
  - ➢ Allows integrity constraint checking before committing a transaction

Problem

- Find expressions ∇S and ΔS such that

  pre(t, S) is equivalent to $(S \ominus \nabla S) \oplus \Delta S$



Current contents of the view

Incremental changes to the view caused by t

- In general, ∇S and ΔS may reference
  - Current state of the database (before t is applied)
    - Including base tables $R_1, \ldots, R_n$, and even S itself
  - Incremental changes to the base tables, $\nabla R_1, \Delta R_1, \ldots, \nabla R_n, \Delta R_n$, to be made by t

"Good" solutions

- Minimality: $\nabla S \ominus S = \varnothing$
  - Do not "over" delete
- Strong minimality: in addition to minimality,

  ∇S min $\Delta S = \varnothing$
  - Do not delete a tuple and then insert it back again
- Why minimality?
  - Rules out "bad" solutions such as $\nabla S = S$, $\Delta S =$ pre(t, S)
  - Simplifies further propagation of deltas
- Does not rule out $\nabla S = S \ominus$ pre(t, S), $\Delta S =$ pre(t, S) $\ominus S$

➢ Need to ensure ∇S and ΔS are easy to evaluate

Change propagation

- A change propagation equation describes how to "bubble up" a delta through a single operator
- For a complex expression, repeatedly apply change propagation equations until all deltas are "bubbled up" to the top of the expression
  - The "bubbles" are the incremental changes
  - The remaining expression corresponds exactly to the current state of the view

Change propagation equations (slide 1)

Most commonly used ones

- $\overbrace{\sigma_p (R \ominus \nabla R)}^{\text{New view}} = \overbrace{\sigma_p (R)}^{\text{Old view}} \ominus \overbrace{\sigma_p (\nabla R)}^{\text{Incremental changes}}$
- $\sigma_p (R \oplus \Delta R) = \sigma_p (R) \oplus \sigma_p (\Delta R)$
- $\pi_A (R \ominus \nabla R) = \pi_A (R) \boxed{\ominus \pi_A (\nabla R \text{ min } R)}$
- $\pi_A (R \oplus \Delta R) = \pi_A (R) \oplus \pi_A (\Delta R)$ Why not just $\ominus \pi_A (\nabla R)$?
- $(R \ominus \nabla R) \times S = (R \times S) \ominus (\nabla R \times S)$
- $(R \oplus \Delta R) \times S = (R \times S) \oplus (\Delta R \times S)$

Change propagation equations (slide 2)

A non-obvious example

- $(R \oplus \Delta R) \ominus S = (R \ominus S) \oplus (\Delta R \ominus (S \ominus R))$
  - Intuition
    - Go ahead and insert ΔR?
    - Almost works; except when S "over" deletes R, it may cancel some effects of ΔR
  - Another intuition
    - $\ominus$ does not maintain negative counts
    - $(S \ominus R)$ recovers these negative counts

Change propagation equations (slide 3)

The only examples where insertion (or deletion) into a base table results in deletion (or insertion, respectively) from the view

- $R \ominus (S \ominus \nabla S) = (R \ominus S) \oplus ((\nabla S \text{ min } S) \ominus (S \ominus R))$
- $R \ominus (S \oplus \Delta S) = (R \ominus S) \ominus \Delta S$

- All other bag algebra operators are monotone!
  - That is, more input means no less output

Change propagation example

View $U = S \oplus T$

- $[(S \ominus \nabla S) \oplus \Delta S] \oplus [(T \ominus \nabla T) \oplus \Delta T]$

bubble up ΔT

- $\{[(S \ominus \nabla S) \oplus \Delta S] \oplus (T \ominus \nabla T)\} \oplus \Delta T$

bubble up ΔS

- $\{(S \ominus \nabla S) \oplus (T \ominus \nabla T)\} \oplus \Delta S \oplus \Delta T$

bubble up ∇T

- $[(S \ominus \nabla S) \oplus T] \ominus (\nabla T \text{ min } T) \oplus \Delta S \oplus \Delta T$

bubble up ∇S

- $(S \oplus T) \ominus (\nabla S \text{ min } S) \ominus (\nabla T \text{ min } T) \oplus \Delta S \oplus \Delta T$

That is, pre(t, U) = $(U \ominus \nabla U) \oplus \Delta U$, where

- $\nabla U = (\nabla S \text{ min } S) \oplus (\nabla T \text{ min } T)$
- $\Delta U = \Delta S \oplus \Delta T$

Minimality patch

- View $U = S \oplus T$
  - $\nabla U = (\nabla S \text{ min } S) \oplus (\nabla T \text{ min } T)$
  - $\Delta U = \Delta S \oplus \Delta T$
- Minimal, but not strongly minimal
  - If x is in both ∇S and ΔT, then x is in both ∇U and ΔU
- Apply minimality "patch"
  - $\nabla_2 U = (U \text{ min } \nabla U) \ominus \Delta U$
  - $\Delta_2 U = \Delta U \ominus (U \text{ min } \nabla U)$
  - Intuition
    - Do not over delete
    - Do not delete something that will be inserted later
    - Do not insert something that was deleted earlier

Recap of change propagation

- Change propagation by hand: complicated, non-deterministic, and may generate deltas that are not minimal!
- Given a view definition U, it would be nice to provide direct definitions for ∇U and ΔU

➢ The paper provides two mutually recursive functions $\nabla(t, U)$ and $\Delta(t, U)$ to compute ∇U and ΔU directly
  - Guarantees strong minimality
  - Exploits the strong minimality assumption to simplify expressions

Examples of $\nabla(t, U)$ and $\Delta(t, U)$

- For a view U of the form $\pi_A(S)$, where S is a subexpression
  - $\nabla(t, U)$ is $\pi_A(\nabla(t, S)) \ominus \pi_A(\Delta(t, S))$
  - $\Delta(t, U)$ is $\pi_A(\Delta(t, S)) \ominus \pi_A(\nabla(t, S))$
- For a view U of the form R, where R is just a base table
  - $\nabla(t, U)$ is ∇R if t modifies R, or $\varnothing$ otherwise
  - $\Delta(t, U)$ is ΔR if t modifies R, or $\varnothing$ otherwise
  - Here we assume ∇R and ΔR are strongly minimal to begin with

➢ Recursively go down the expression tree, until we hit the leaves (base tables)

Deriving $\nabla(t, U)$ and $\Delta(t, S)$

Example: consider a view U of the form $\pi_A(S)$

$$
\begin{aligned}
&\text{pre}(t, \pi_A(S))\\
&= \pi_A(\text{pre}(t, S))\\
&= \pi_A((S \ominus \nabla(t, S)) \oplus \Delta(t, S))\\
&= \pi_A(S \ominus \nabla(t, S)) \oplus \pi_A(\Delta(t, S))\\
&= (\pi_A(S) \ominus \pi_A(\nabla(t, S) \min S)) \oplus \pi_A(\Delta(t, S))\\
&= (\pi_A(S) \ominus \pi_A(\nabla(t, S))) \oplus \pi_A(\Delta(t, S))
\end{aligned}
$$

bubble up $\Delta(t, S)$

bubble up $\nabla(t, S)$

simplification based on minimality assumption

Finally, applying the minimality patch, we get:

$\nabla(t, U) = \pi_A(\nabla(t, S)) \ominus \pi_A(\Delta(t, S))$

$\Delta(t, U) = \pi_A(\Delta(t, S)) \ominus \pi_A(\nabla(t, S))$

Aggregate functions

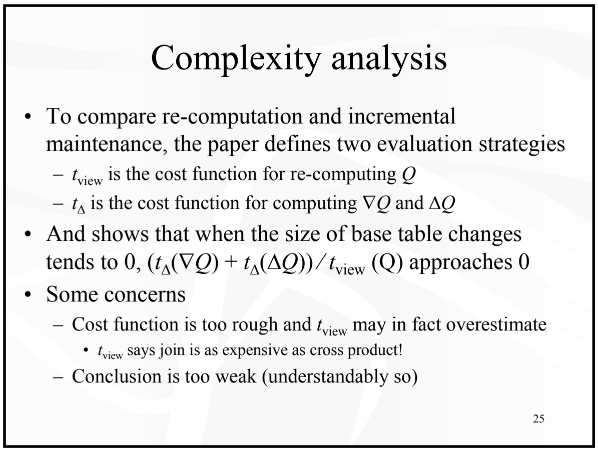
- The paper does not present a complete solution
  - Aggregates are not modeled in bag algebra
  - No GROUP-BY is considered
  - Aggregate maintenance is not handled in the same change propagation framework
  - ➢One aggregate operation is allowed at the very end

SUM, COUNT, AVG, STDEV, …

- Can be defined by expression $\varphi(\sum_{f1}, \sum_{f2}, \ldots, \sum_{fn})$, where each $\sum_f(R)$ sums up $f(x)$ for all x in R
  - SUM = $\sum_{id} = \sum_{x \in R} x$
  - COUNT = $\sum_1 = \sum_{x \in R} 1$
  - AVG = $\sum_{id} / \sum_1$
- Each $\sum_f$ can be materialized and maintained incrementally (assuming minimality of deltas)
  - $\sum_f((R \ominus \nabla R) \oplus \Delta R) = \sum_f(R) - \sum_f(\nabla R) + \sum_f(\Delta R)$

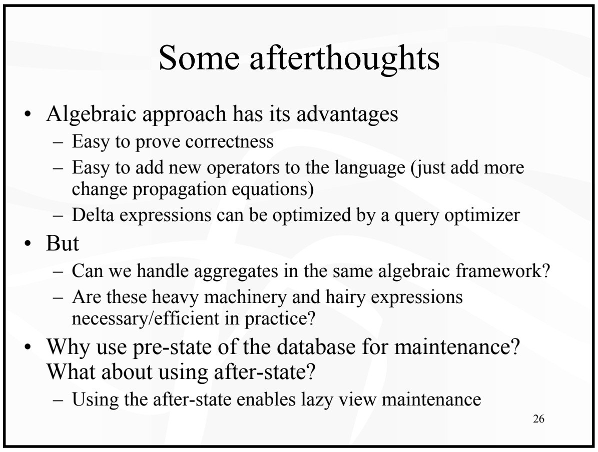
MIN, MAX

- Insertions
  - No problem; simply compare and keep the current MIN/MAX
- Deletions
  - No effect if the current MIN/MAX is not deleted
  - Problematic if the current MIN/MAX is deleted; need to re-compute
- In general, re-computation is required if a transaction deletes the current MIN/MAX and does not insert a new MIN/MAX



Complexity analysis

- To compare re-computation and incremental maintenance, the paper defines two evaluation strategies
  - t_{view} is the cost function for re-computing Q
  - t_{Δ} is the cost function for computing ∇Q and ΔQ
- And shows that when the size of base table changes tends to 0, $(t_{\Delta}(\nabla Q) + t_{\Delta}(\Delta Q)) / t_{\text{view}}$ (Q) approaches 0
- Some concerns
  - Cost function is too rough and t_{view} may in fact overestimate
    - t_{view} says join is as expensive as cross product!
  - Conclusion is too weak (understandably so)



Some afterthoughts

- Algebraic approach has its advantages
  - Easy to prove correctness
  - Easy to add new operators to the language (just add more change propagation equations)
  - Delta expressions can be optimized by a query optimizer
- But
  - Can we handle aggregates in the same algebraic framework?
  - Are these heavy machinery and hairy expressions necessary/efficient in practice?
- Why use pre-state of the database for maintenance? What about using after-state?
  - Using the after-state enables lazy view maintenance