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
  – $\sigma_p(S)$ (selection), $\pi_A(S)$ (projection), $\times$ (cross product)
    • Above three are the most commonly used
  – $\oplus$ (additive union), $\ominus$ (monus), min (minimum intersection), max (maximum union)
  – $e$ (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
  – $\text{count}((r, s), R \times S) = \text{count}(r, R) \cdot \text{count}(s, S)$
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:** $R \setminus S$
  - count($x$, $R \setminus S$) = count($x$, $R$) - count($x$, $S$); or 0 if count($x$, $R$) < count($x$, $S$)
  - Example: {2 apples, 2 bananas} $\setminus$ {3 apples, 1 banana} = {1 banana}

- **Duplicate elimination:** $\epsilon(S)$
  - count($x$, $\epsilon(S)$) = 1; or 0 if $x$ is not in $S$ at all
  - Example: $\epsilon$({2 apples, 2 bananas}) = {1 apple, 1 banana}

Bag algebra operators (slide 3)

- **Minimum intersection:** $R \cap S$
  - count($x$, $R \cap S$) = min(count($x$, $R$), count($x$, $S$))
  - Example: {2 apples} $\cap$ {3 apples} = {2 apples}
  - Can you define it using the other operators?
  - $R \cap S = R \setminus (R \setminus S)$

- **Maximum union:** $R \cup S$
  - count($x$, $R \cup S$) = max(count($x$, $R$), count($x$, $S$))
  - Example: {2 apples} $\cup$ {3 apples} = {3 apples}
  - Can you define it using the other operators?
  - $R \cup S = R \oplus (S \setminus R) = (R \oplus S) \setminus (R \cap S)$

Describing changes to base tables

- A transaction $t$ modifies $R_1, \ldots, R_n$ in one atomic step
  - $R_i \leftarrow (R_i \setminus \Delta R_i) \oplus \Delta R_i$
  - $\cdots$
  - $R_n \leftarrow (R_n \setminus \Delta R_n) \oplus \Delta R_n$

- $\nabla R_i$ contains
  - Tuples deleted by $t$ from $R_i$
  - Old contents of the $R_i$ tuples updated by $t$

- $\Delta R_i$ 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 $S (R_1 \setminus \nabla R_1) \oplus \Delta R_1, \ldots, (R_n \setminus \nabla R_n) \oplus \Delta R_n)$
  - 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 $\nabla S$ and $\Delta S$ such that $\text{pre}(t, S)$ 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, $\nabla S$ and $\Delta S$ 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: $\nabla S \ominus S = \emptyset$
  - Do not “over” delete

- Strong minimality: in addition to minimality, $\nabla S \cap \Delta S = \emptyset$
  - Do not delete a tuple and then insert it back again

- Why minimality?
  - Rules out “bad” solutions such as $\nabla S = S, \Delta S = \text{pre}(t, S)$
  - Simplifies further propagation of deltas

- Does not rule out $\nabla S = S \ominus \text{pre}(t, S), \Delta S = \text{pre}(t, S) \ominus S$
  - Need to ensure $\nabla S$ and $\Delta S$ 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:

\[
\begin{align*}
\sigma_p (R \ominus \nabla R) &= \sigma_p (R) \ominus \sigma_p (\nabla R) \\
\sigma_p (R \oplus \Delta R) &= \sigma_p (R) \oplus \sigma_p (\Delta R) \\
\pi_A (R \ominus \nabla R) &= \pi_A (R) \ominus \pi_A (\nabla R \min A) \\
\pi_A (R \oplus \Delta R) &= \pi_A (R) \oplus \pi_A (\Delta R) \quad \text{Why not just } \ominus \pi_A (\nabla R)? \\
(R \ominus \nabla R) \times S &= (R \times S) \ominus (\nabla R \times S) \\
(R \oplus \Delta R) \times S &= (R \times S) \oplus (\Delta R \times S)
\end{align*}
\]

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 \(\Delta R\)?
- Almost works; except when \(S\) “over” deletes \(R\), it may cancel some effects of \(\Delta R\).
- Another intuition
  - \(\ominus\) does not maintain negative counts
  - \((S \ominus R)\) recovers these negative counts.

Change propagation example

View \(U = S \ominus T\)

\[
\begin{align*}
[S \ominus \nabla S \ominus \Delta S] @ [(T \ominus \nabla T) @ \Delta T] & \quad \text{bubble up } \Delta T \\
[(S \ominus \nabla S \ominus \Delta S) @ (T \ominus \nabla T)] @ \Delta T & \quad \text{bubble up } \Delta S \\
(S \ominus \nabla S) @ (T \ominus \nabla T) & \quad \ominus \Delta S @ \Delta T \\
(S \ominus \nabla S) \ominus T & \quad \ominus \Delta S @ \Delta T \\
(S \ominus \nabla S) @ (T \ominus \nabla T) & \quad \ominus \Delta S @ \Delta T \\
(S \ominus \nabla S) \ominus T & \quad \ominus \Delta S @ \Delta T
\end{align*}
\]

That is, \(\text{pre}(t, U) = (U \ominus \nabla U) \oplus \Delta U\), where

- \(\nabla U = (\nabla S \ominus S) \ominus (\nabla T \ominus T)\)
- \(\Delta U = \Delta S \ominus \Delta T\)

Minimality patch

- View \(U = S \ominus T\)
  - \(\nabla U = (\nabla S \ominus S) \ominus (\nabla T \ominus T)\)
  - \(\Delta U = \Delta S \ominus \Delta T\)
- Minimal, but not strongly minimal
  - If \(x\) is in both \(\nabla S\) and \(\Delta T\), then \(x\) is in both \(\nabla U\) and \(\Delta U\).
- Apply minimality “patch”
  - \(\nabla U = ((U \ominus \nabla U) @ \Delta U)\)
  - \(\Delta U = \Delta U \ominus (\nabla U \ominus \Delta U)\)
- 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 $\nabla U$ and $\Delta U$

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

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

- For a view $U$ of the form $\pi_d(S)$, where $S$ is a subexpression
  - $V(t, U)$ is $\pi_d(V(t, S)) \oplus \pi_d(M(t, S))$
  - $\Delta(t, U)$ is $\pi_d(M(t, S)) \oplus \pi_d(V(t, S))$
- For a view $U$ of the form $R$, where $R$ is just a base table
  - $V(t, U)$ is $VR$ if $t$ modifies $R$, or $\emptyset$ otherwise
  - $\Delta(t, U)$ is $AR$ if $t$ modifies $R$, or $\emptyset$ otherwise
  - Here we assume $VR$ and $AR$ 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 $\pi_d(S)$

\[
\pre(t, \pi_d(S)) = \pi_d(\pre(t, S)) = \pi_d((S \ominus \nabla(t, S)) \oplus \Delta(t, S))
\]

bubble up $\Delta(t, S)$

simplification based on minimality assumption

Finally, applying the minimality patch, we get:

\[
\nabla(t, U) = \pi_d(\nabla(t, S)) \oplus \pi_d(M(t, S))
\]

\[
\Delta(t, U) = \pi_d(M(t, S)) \oplus \pi_d(V(t, S))
\]

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 $f(\sum_1, \sum_2, \ldots, \sum_n)$, where each $\sum_j(R)$ sums up $f(x)$ for all $x$ in $R$
  - $\sum_1$ = $\sum_{x \in R} x$
  - $\sum_1$ = $\sum_{x \in R} 1$
  - $\sum_1$ = $\sum_{x \in R} x$
- Each $\sum_j$ can be materialized and maintained incrementally (assuming minimality of deltas)
  - $\sum_j((R \ominus VR) \oplus DR) = \sum_j(R) - \sum_j(VR) + \sum_j(DR)$

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
  – $t_{\text{view}}$ is the cost function for re-computing $Q$
  – $t_{\Delta}$ is the cost function for computing $\nabla Q$ and $\Delta Q$
• 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 $t_{\text{view}}$ may in fact overestimate
    • $t_{\text{view}}$ 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