Incremental Maintenance for Non-Distributive Aggregate Functions

Themis Palpanas (U of Toronto)
Richard Sidle
Bobbie Cochrane
Hamid Pirahesh

work done at IBM Almaden Research Center
Motivation

- large amounts of data stored in databases
  - expensive OLAP queries, but with nice properties:
    - based on same set of tables
    - perform similar aggregations
- can efficiently support such queries with
  - Automatic Summary Tables (ASTs)
    - precomputed once, used many times
    - answer complex queries fast
- must maintain ASTs when base tables change
  - inserts, updates, deletes
Motivation (cont’d)

- AST
- AST definition
- base tables
- insert/update/delete
Problem Statement

given ASTs with aggregate functions
- distributive
  - SUM, COUNT
- non-distributive
  - STDDEV, CORRELATION, REGRESSION, MIN/MAX, XMLAGG, ...

when base tables change
- incrementally maintain affected ASTs

efficient maintenance of ASTs with non-distributive aggregate functions
Outline

- Current Approach
- Our Solution
- Experimental Evaluation
- Related Work
- Conclusions
Current Approach

Propagate phase

vars

combine old and new values

Apply phase

AST

definition

AST

insert/update/delete

delta
Current Approach (cont’d)

- works for distributive
  - SUM, COUNT

- does not work for non-distributive
  - STDDEV, CORRELATION, REGRESSION
  - MIN/MAX
  - XMLAGG

- need new way to deal with these functions
Our Solution

- selective recomputation
  - no longer enough to compute delta
  - must recompute some aggregation groups

- minimize work to be done
  - choose which groups to recompute
  - optimize query plan
Our Solution (cont’d)

Propagate phase

delta

recompute affected groups

combine old and new values

Apply phase

AST
definition

base tables

insert/update/delete
Our Solution (cont’d)

the 5 steps

1. compute new aggregate values
2. change column derivation
3. recompute only affected groups
4. eliminate unnecessary operations
5. optimize for special cases
Initial Query Plan

Query Graph Model (QGM)

UDI

LOJ

prop

AST
1. Compute New Aggregate Values

- compute delta for distributive functions
- recompute non-distributive functions
- get those values only for affected groups

- duplicate computation for distributive functions!
2. Change Column Derivation

- change column derivation
- rewrite phase projects out unused columns

- entire AST gets recomputed!
3. Recompute Affected Groups

push join predicate down in AST
- only affected groups are recomputed

special rules for super-aggregates
3. Recompute Affected Groups

- special treatment for ASTs with super-aggregates
  - predicates not pushdownable
  - caution not to compute totals of totals
- build special join predicate
  - ensure correct aggregations
- change rewrite rules
  - allow predicate pushdown through super aggregates
  - applicable only for special join predicate
4. Remove Unnecessary Operations

- outerjoin not always needed
- when changes are only inserts
  - reroute columns from propagate phase through AST
  - remove outerjoin operator
- same for updates not referencing AST grouping columns and predicates
5. Optimize for Special Cases

- recomputation step not needed when
  - only insertions and only MIN/MAX functions
    - build predicate in apply phase
    - check if new min/max should replace old values
  - only deletions referring only to grouping columns of AST
    - can only cause entire groups to be deleted
    - handled in apply phase
Experimental Evaluation

- prototype implementation in IBM DB2 UDB
- star schema database
  - sales of products over 5 year time period
  - fact table: 10 million tuples
- AST with non-distributive aggregate function
  - 240,000 tuples
- workload simulates nightly updates
  1. add/delete data for first day of month
  2. add/delete data for second day of month
  3. add/delete data for full month
### Experimental Evaluation (cont’d)

<table>
<thead>
<tr>
<th></th>
<th>workload 1</th>
<th>workload 2</th>
<th>workload 3</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>incremental</strong></td>
<td>286</td>
<td>294</td>
<td>420</td>
</tr>
<tr>
<td>full refresh</td>
<td>699</td>
<td>702</td>
<td>692</td>
</tr>
</tbody>
</table>

#### deletions require 40-60% of full refresh time

<table>
<thead>
<tr>
<th></th>
<th>workload 1</th>
<th>workload 2</th>
<th>workload 3</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>incremental</strong></td>
<td>3</td>
<td>n/a</td>
<td>31</td>
</tr>
<tr>
<td>full refresh</td>
<td>699</td>
<td>702</td>
<td>692</td>
</tr>
</tbody>
</table>

#### optimized deletions require 1-4% of full refresh time
Experimental Evaluation (cont’d)

<table>
<thead>
<tr>
<th></th>
<th>workload 1</th>
<th>workload 2</th>
<th>workload 3</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>incremental</strong></td>
<td>151</td>
<td>158</td>
<td>180</td>
</tr>
<tr>
<td><strong>full refresh</strong></td>
<td>702</td>
<td>702</td>
<td>721</td>
</tr>
</tbody>
</table>

◆ insertions updates require 20-25% of full refresh time
Related Work

- incremental view maintenance
  - differential refresh algorithms
  - deferred incremental maintenance
- views with aggregation
Conclusions

- incremental maintenance for ASTs with non-distributive aggregate functions
  - support MIN/MAX, STDDEV, CORRELATION, REGRESSION, XMLAGG, ...
- efficient selective recomputation
  - recompute only affected groups
  - optimize query plan
  - customize for special cases
- significant performance improvements