Warehouses & Virtual Databases offer integrated views

Focus: Sources are relational DBs. Integration specified by distributed view definition(s). Clients issue queries on views.
Virtual View -> Mediator needs Distributed Query Processor
Materialized view (warehouse) -> Mediator also needs storage & Incremental View Maintenance

Distributed Query Processing in Mediators

Mediators as providers of view-based virtual views over distributed data

Translate subplans to SQL
+ How will the join happen?
- What if the source offers multiple data services instead of JDBC access? (besides CSE232 material)
Join types

- **Mediator-based Join**
  - Ship results of queries at mediator

- **Parameterized Join**
  - Right subquery is enhanced with selection on join attribute
  - For each join value of left hand side, execute another right subquery

- **Data Ship Join**
  - Insert the result of left hand side (lhs) in the db of right hand side (rhs).
  - Execute join at db of right hand side

- **Semijoin Reduction Join**
  - Send rhs parameters to lhs
  - (Data ship alike variation) Lhs sends to rhs the semijoin of its subquery with the parameters set.
  - Execute join at db of rhs
  - Also, variation that looks like mediator-based join

Virtual Views Vs Materialized Views

<table>
<thead>
<tr>
<th>View kind</th>
<th>CREATE VIEW V AS</th>
<th>CREATE MATERIALIZED VIEW V AS</th>
</tr>
</thead>
<tbody>
<tr>
<td>FROM R</td>
<td>SELECT G, SUM(A) AS S FROM R GROUP BY G</td>
<td>SELECT G, SUM(A) AS S FROM R GROUP BY G</td>
</tr>
</tbody>
</table>

Upon Updating R

- Database does nothing
- (Ideally) Database must refresh V to reflect changes on R

Upon Querying V

- SELECT * FROM V WHERE G = 5
- \( \sigma_{G=5} \) \( V \)
- \( \sigma_{G=5} \) \( V\) \( G,\text{SUM}(A)=S \)

Upon Optimizing & running R

- \( \sigma_{G=5} \) \( V \)
- \( \sigma_{G=5} \) \( V \)

Recompute Vs Incremental (Materialized View) Maintenance – Informal Example

<table>
<thead>
<tr>
<th>CREATE MATERIALIZED VIEW V AS</th>
</tr>
</thead>
<tbody>
<tr>
<td>SELECT G, SUM(A) AS S FROM R</td>
</tr>
<tr>
<td>GROUP BY G</td>
</tr>
</tbody>
</table>

At the end of the transaction, we want V to reflect the new state of R

Option 1: Delete and Recompute V

Option 2: Incrementally maintain V

\( \Delta R^* \): the set of tuples inserted in R (obtained by log or other mechanism)

At the end of the transaction, we want V to reflect the new state of R

UPDATE V

SET S = S +

(\( \sigma_{\Delta R^*} \) \( G = V.G \))

WHERE \( V.G \) IN (SELECT G FROM \( \Delta R^* \));

INSERT INTO V

(\( \sigma_{\Delta R^*} \) \( A.S FROM \( \Delta R^* \))

WHERE NOT \( G \) IN (SELECT G FROM V)

GROUP BY G);
Capturing IVM as computation of \( \Delta V^+, \Delta V^- \)

WITH RGplus AS
(SELECT G, SUM(A) AS S FROM R GROUP BY G),
RGminus AS
(SELECT G, SUM(A) AS S FROM R GROUP BY G),
RGnet AS
(SELECT choose(p.G, m.G) AS G, n0(p.S) - n0(m.S) AS S FROM RGplus p FULL OUTER JOIN RGminus m ON p.G = m.G)

\( \Delta V^+ \) AS
(SELECT * FROM V WHERE G IN (SELECT G FROM RGnet))

\( \Delta V^- \) AS
(SELECT r.G AS G, n0(V.S) + r.S AS S FROM (V RIGHT OUTER JOIN RGnet AS r ON V.G = r.G)

\text{choose}(a, b) \text{ returns } a \text{ if } a \text{ is NOT NULL, returns } b \text{ if } a \text{ is NULL }

n0(a) \text{ returns } a \text{ if } a \text{ is NOT NULL, 0 otherwise}
IVM: Self-maintaining version (not always possible)

Problem: Find efficient view updates

\[ \Delta V^+ = f(\Delta R_1^*, \ldots, \Delta R_n^*, V^0) \]

\[ \Delta V^- = f(\ldots) \rightarrow V^1 = V(R_1^1, \ldots, R_n^1) \]

View

\[ V^0 = V(R_1^0, \ldots, R_n^0) \]

\[ V^1 = V(R_1^1, \ldots, R_n^1) \]

Database tables

From logs or intercepted by triggers

Table Updates

\[ \Delta R_1^*, \ldots, \Delta R_n^* \]

\[ \Delta R_1^1, \ldots, \Delta R_n^1 \]

Basic IVM Algorithm: Compose operator IVM rules

Example (wlog deferred, i.e., R means R^1 and S means S^1)

- Rule for \( V = R \bowtie S \)
  - \( \Delta V^+ = (\Delta R^* \bowtie S) \cup (R \bowtie \Delta S^*) - (\Delta R^* \bowtie \Delta S^*) \)
  - \( \Delta V^- = \ldots \)
- Rule for \( V = \sigma_c R \)
  - \( \Delta V^+ = \sigma_c \Delta R^* \)
  - \( \Delta V^- = \ldots \)
- Composition of rules leads to solutions for \( V = T \bowtie \sigma_{A=S} W \)
  - \( \Delta V^+ = \ldots \)
  - \( \Delta V^- = \ldots \)
- May rewrite initial expression

IVM with Caching

- May associate intermediate views (caches) with subexpressions
- Bottom-up: From updating caches to reaching the materialized view
- Caches will typically needed indices
- Caches may or may not pay off as they incur cost for maintaining them (and their indices)
Generalizations

- Multiple views
  - Self maintenance may involve a view utilizing the other views in its computation
- Genuine updates
  - Not simulated via insertions/deletions
- Insertions, deletions, updates on tables and views expressed as DML statements

Comparisons

<table>
<thead>
<tr>
<th>Materialized View</th>
<th>Virtual View</th>
</tr>
</thead>
<tbody>
<tr>
<td>High query performance</td>
<td>No need for yet another database</td>
</tr>
<tr>
<td>Queries not visible outside warehouse</td>
<td>More up-to-date data</td>
</tr>
<tr>
<td>Local processing at sources unaffected</td>
<td>Depending on specifics of IVM</td>
</tr>
<tr>
<td>Can operate when sources unavailable</td>
<td>Query needs can be unknown</td>
</tr>
<tr>
<td>Extra information at warehouse</td>
<td>Only query interface needed at sources</td>
</tr>
</tbody>
</table>
  - Modify, summarize (store aggregates) |
  - Add historical information |
=> Lower Total Cost of Ownership