Materialized Views

March 26, 2018
CREATE VIEW salesSinceLastMonth AS
SELECT l.*
FROM lineitem l, orders o
WHERE l.orderkey = o.orderkey
AND o.orderdate > DATE('2015-03-31')

SELECT partkey FROM salesSinceLastMonth
ORDER BY shipdate DESC LIMIT 10;

SELECT suppkey, COUNT(*)
FROM salesSinceLastMonth
GROUP BY suppkey;

SELECT partkey, COUNT(*)
FROM salesSinceLastMonth
GROUP BY partkey;
CREATE VIEW salesSinceLastMonth AS
SELECT l.*
FROM lineitem l, orders o
WHERE l.orderkey = o.orderkey
    AND o.orderdate > DATE('2015-03-31')

SELECT partkey FROM salesSinceLastMonth
ORDER BY shipdate DESC LIMIT 10;

SELECT partkey FROM
    (SELECT l.*
     FROM lineitem l, orders o
     WHERE l.orderkey = o.orderkey
         AND o.orderdate > DATE('2015-03-31')
     ) AS salesSinceLastMonth
ORDER BY shipdate DESC LIMIT 10;
Views

• … contain and abstract complex concepts.
  • Complex query patterns can be given a shorthand.
  • It’s easier to change view logic “in the background”
• … act as normal relations.
  • View references can be expanded inline into nested subqueries.
• Updates are trickier…. 
View Updates

What happens when we Insert Into/Update a view?
View Updates

```
UPDATE salesSinceLastMonth
    SET statusCode = 'q';
WHERE orderkey = 22;
```

Rows in `salesSinceLastMonth` correspond 1-1 with rows in `lineitem`. Update `lineitem`!
View Updates

```sql
INSERT INTO salesSinceLastMonth
    (orderkey, partkey, suppkey, ...)
VALUES
    (22, 99, 42, ...);

Lots of problems...
- What if order # 22 doesn’t exist?
- How does the insertion interact with sequences
  (e.g., lineitem.lineno)
```
View Updates

**Solution 1:** Data Integration (CSE 636)

**Solution 2:** INSTEAD OF triggers
CREATE TRIGGER salesSinceLastMonthInsert
INSTEAD OF INSERT ON salesSinceLastMonth
REFERENCING NEW ROW AS newRow
FOR EACH ROW
  IF NOT EXISTS (  
    SELECT * FROM ORDERS  
    WHERE ORDERS.orderkey = newRow.orderKey  
  ) THEN  
    INSERT INTO ORDERS(orderkey)  
    VALUES (orderkey)  
  END IF;  
  INSERT INTO LINEITEM VALUES newRow;  
END FOR;
Can we use views for anything else?
Materialization

Views exist to be queried frequently

Pre-compute and save the view’s contents!
(like an index)
Materialization Challenges

• How do we maintain the views as data changes?

• What if the view is not explicitly referenced?

• What views should be materialized?
Delta Queries

• If D is your Database and Q is your Query:
  • Q(D) is the result of your query on the database.
  • Let’s say you make a change ΔD (Insert tuple)
    • Q(D+ΔD) is the new result
  • If we have Q(D), can we get Q(D+ΔD) faster?
    • Analogy to Sum: \{34, 29, 10, 15\} + \{12\} (88+12)
Query Rewriting

CREATE MATERIALIZED VIEW salesSinceLastMonth AS
  SELECT l.*
  FROM lineitem l, orders o
  WHERE l.orderkey = o.orderkey
    AND o.orderdate > DATE('2015-03-31')

SELECT l.partkey
FROM lineitem l, orders o
WHERE l.orderkey = o.orderkey
  AND o.orderdate > DATE('2015-03-31')
ORDER BY l.shipdate DESC
LIMIT 10;

We can use a materialized view to speed the query up
# Query Rewriting

<table>
<thead>
<tr>
<th>View Query</th>
<th>User Query</th>
</tr>
</thead>
<tbody>
<tr>
<td>SELECT ( L_V )</td>
<td>SELECT ( L_Q )</td>
</tr>
<tr>
<td>FROM ( R_V )</td>
<td>FROM ( R_Q )</td>
</tr>
<tr>
<td>WHERE ( C_V )</td>
<td>WHERE ( C_Q )</td>
</tr>
</tbody>
</table>

When are we allowed to rewrite this query?
Query Rewriting

View Query

SELECT $L_V$
FROM $R_V$
WHERE $C_V$

User Query

SELECT $L_Q$
FROM $R_Q$
WHERE $C_Q$

$R_V \subseteq R_Q$
All relations in the view are part of the query join

$C_Q = C_V \land C'$
The view condition is weaker than the query condition

$\text{attrs}(C') \cap \text{attrs}(R_V) \subseteq L_V$
$L_Q \cap \text{attrs}(R_V) \subseteq L_V$
The view doesn’t project away needed attributes
### Query Rewriting

<table>
<thead>
<tr>
<th>View Query</th>
<th>User Query</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>SELECT L_v</code></td>
<td><code>SELECT L_Q</code></td>
</tr>
<tr>
<td><code>FROM R_v</code></td>
<td><code>FROM R_Q</code></td>
</tr>
<tr>
<td><code>WHERE C_v</code></td>
<td><code>WHERE C_Q</code></td>
</tr>
</tbody>
</table>

What does the query rewrite to?
## Query Rewriting

<table>
<thead>
<tr>
<th>View Query</th>
<th>User Query</th>
</tr>
</thead>
<tbody>
<tr>
<td>SELECT $L_V$ FROM $R_V$ WHERE $C_V$</td>
<td>SELECT $L_Q$ FROM $R_Q$ WHERE $C_Q$</td>
</tr>
</tbody>
</table>

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>SELECT $L_Q$ FROM $(R_Q-R_V), \text{ VIEW}$ WHERE $C_Q$</td>
<td></td>
</tr>
</tbody>
</table>
Materialized Views

When the base data changes, the view needs to be updated
View Maintenance

VIEW ← Q(D)
View Maintenance

WHEN $D \leftarrow D + \Delta D$ DO:

$\text{VIEW} \leftarrow Q(D + \Delta D)$

Re-evaluating the query from scratch is expensive!
View Maintenance

\[
\text{WHEN } D \leftarrow D + \Delta D \text{ DO:} \\
\text{VIEW } \leftarrow \text{VIEW} + \Delta Q(D, \Delta D)
\]

(ideally) Smaller & Faster Query

(ideally) Fast “merge” operation.
Intuition

$$D = \{1, 2, 3, 4\} \quad \Delta D = \{5\}$$

$$Q(D) = \text{SUM}(D)$$

$$Q(D + \Delta D) \sim O(|D| + |\Delta D|)$$

$$\text{VIEW} + \text{SUM}(\Delta D) \sim O(|\Delta D|)$$
Intuition

\[ R = \{1, 2, 3\}, \quad S = \{5, 6\} \quad \Delta R = \{4\} \]

\[ Q(R, S) = \text{COUNT}(R \times S) \]

\[ Q(R+\Delta R, S) \sim O\left( (|R| + |\Delta R|) \times |S| \right) \]

\[ \text{VIEW} + \text{COUNT}(|\Delta R| \times |S|) \sim O(|\Delta R| \times |S|) \]
Intuition

+ \sim U

* \sim X

Are these kinds of patterns common?
Rings/Semirings

This kind of pattern occurs frequently.

**Semiring**: < S, +, x, S₀, S₁ >

Any set of ‘things’ S such that...

\[
\begin{align*}
S_i + S_j &= S_k \\
S_i \times S_j &= S_k \\
S_i \times (S_j + S_k) &= (S_i \times S_j) + (S_j \times S_k)
\end{align*}
\]

Closed

\[
\begin{align*}
S_i + S_0 &= S_i \\
S_i \times S_1 &= S_i \\
S_i \times S_0 &= S_0
\end{align*}
\]

Additive & Multiplicatively "zeroes"

Distributive
Rings/Semirings

Ring : \(< S, +, x, S_0, S_1, - >\)

Any semiring where every element has an additive inverse...

\[ S_i + (-S_i) = S_0 \]
THE TANGENT ENDS NOW