**Sales Q6**

To measure performance of sales query 6 for Milestone 4, I created a database with the following characteristics:

* 1,000 customers in all 50 states (20 per state)
* 1,000 products in 100 categories (10 per category)
* Only 10 customers appear in sales table
* Only 10 products appear in sales table
* Only 10 states appear in sales table
* 5 million sales

Prior to precomputations (with no indices):

Explain Analyze execution time: 37757.274 ms

Direct run execution time: 32 secs

After precomputations (with no indices):

Explain Analyze execution time: 1179.458 ms

Direct run execution time: 1 secs

Added precomputations:

-- top category (equivalent to top\_category view)

-- top customer (equivalent to top\_customer view)

-- sales by product and category (equivalent to q4 view)

CREATE MATERIALIZED VIEW top\_category\_m AS

SELECT category\_id,

sum(dollar\_value) AS dollar\_value

FROM q5

GROUP BY category\_id

ORDER BY dollar\_value DESC

LIMIT 20;

CREATE MATERIALIZED VIEW top\_customer\_m AS

SELECT customer\_id

FROM q1

ORDER BY dollar\_value DESC

LIMIT 20;

CREATE MATERIALIZED VIEW sales\_by\_prod\_cust AS

SELECT c.customer\_id,

c.customer\_name,

p.product\_id,

COALESCE(sum(s.quantity), 0) AS quantity\_sold,

COALESCE(sum(s.quantity \* s.price), 0.0) AS dollar\_value,

c.state\_id,

p.category\_id

FROM (customer c CROSS JOIN product p) LEFT JOIN sale s

ON c.customer\_id = s.customer\_id AND p.product\_id = s.product\_id

GROUP BY c.customer\_id, p.product\_id

ORDER BY c.customer\_id, dollar\_value DESC;

Query Modifications:

-- modify q6 to utilize precomputed tables

CREATE VIEW q6m AS

SELECT ca.category\_id, cu.customer\_id,

coalesce (SUM (q.quantity\_sold), 0) AS quantity\_sold,

coalesce (SUM (q.dollar\_value), 0.0) AS dollar\_value

FROM (top\_customer\_m cu CROSS JOIN top\_category\_m ca) LEFT JOIN sales\_by\_prod\_cust q

ON q.customer\_id = cu.customer\_id AND q.category\_id = ca.category\_id

GROUP BY ca.category\_id, cu.customer\_id;

Sales Precomputation Notes/Discussion:

Utilizing precomputed tables for top customers, top categories, and sales by product and customer results in much faster execution times for queries 4, 5, and 6. Maintaining the top customer and top category tables should not be too burdensome, and the top 20 customers and categories are likely to be used often for various types of queries, or even as standalone queries.

Maintaining the sales by product and customer will require updating the table upon every addition, update, or deletion to the sale table, and every addition or deletion to the customer and product tables. Since a query (i.e. q4) was written specifically to calculate sales by customer and product, and both queries 5 and 6 benefit from precomputation, it is probably a safe assumption that the overhead associated with maintaining the table will be worthwhile.

**Cats Q5 (Weighted Likes)**

To measure performance of cats query 5 for Milestone 4, I created a database with the following characteristics:

* 5,000 user table rows
* 3,000 video table rows
* 20,000 friend table rows
* 10,000 likes table rows
* 20,000 watch table rows
* 20,000 login table rows
* 5,000 suggestion table rows

Prior to precomputations (with no indices):

Explain Analyze execution time: 22.270 ms

Direct run execution time: 146 msec

After precomputations (with no indices):

Explain Analyze execution time: 4309.222 ms

Direct run execution time: 4 secs

After precomputations (with index on weightedMykindLikes.uid):

Explain Analyze execution time: 10.721 ms

Direct run execution time: 100 msecs

Added precomputations:

-- weighted my kind of likes (equivalent to weightedMykindLikes view)

CREATE MATERIALIZED VIEW weightedmykindlikes\_m (uid, vid, verdict) AS

SELECT u.user\_id as uid, l.video\_id AS vid, log(1+i.prod) AS verdict

FROM cats.user u, inner\_product i, likes l

WHERE u.user\_id = i.x and l.user\_id = i.y;

Query Modifications:

Replace overallLikes with precomputed weightedMykindLikes\_m in the original Overall\_best query.

Cats Precomputation Notes/Discussion:

Addition of precomputed weightedmykindlikes table without an index on uid actually increases execution time for query 5 on my data, due to the need to sequentially scan the (large) weightedmykindlikes table, which has a number of rows roughly equal to the total number of users times the total number of likes. Adding an index on uid to the precomputed weightedmykindlikes table reduces the execution time to lower than the initial query, but maintaining the weightedmykindlikes table may not be practical, as the time required to recalculate the weightedmykindlikes table is significant (on my database, execution of explain analyze is roughly 1 minute, and execution of a direct run would be much longer due to the time required to output the query results). Therefore, maintaining the precomputed table is a tradeoff: in order for it to be practical to update the precomputed table every time a video is liked, the relative frequency with which users like videos would need to be quite low compared to the frequency with users are suggested a list of videos based on the weighted rankings. On the other hand, if users would be satisfied with recommendations based on likes info that is not completely current, the weightedmykindlikes table could be recalculated at some specified time interval (hourly, for example).