Paul de Fusco

3/12/2017

Homework 4

NB: I refreshed the cache every time I ran a new scenario, multiple times if necessary

SALES CUBE:

**First Scenario**: create materialized view in place of underlying views q1, q4, q5 only

1. create materialized views

create materialized view sales.q1view

as

SELECT c.customer\_id, s.product\_id,

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

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

FROM sales.customer c LEFT JOIN sales.sale s ON c.customer\_id = s.customer\_id

GROUP BY c.customer\_id, s.product\_id;

CREATE materialized VIEW sales.q4view 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,

-- needed later:

c.state\_id, p.category\_id

FROM (sales.customer c CROSS JOIN sales.product p) LEFT JOIN sales.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;

CREATE materialized VIEW sales.q5view AS

--reusing Q4

SELECT s.state\_id, c.category\_id,

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

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

FROM (sales.state s CROSS JOIN sales.category c)

LEFT JOIN sales.q4 q ON s.state\_id = q.state\_id AND c.category\_id = q.category\_id

GROUP BY s.state\_id, c.category\_id;

2) Recreate existing views in top layer to use new underlying materialized views

drop view sales.top\_customer\_values

drop view sales.top\_category\_values

drop view sales.all\_top\_customers

drop view sales.all\_top\_categories

drop view sales.q6\_all

CREATE VIEW sales.top\_customer\_values AS

SELECT DISTINCT dollar\_value

FROM sales.q1view

ORDER BY dollar\_value DESC

LIMIT 20;

CREATE VIEW sales.all\_top\_customers AS

SELECT customer\_id

FROM sales.q1view

WHERE dollar\_value IN (SELECT dollar\_value FROM sales.top\_customer\_values);

CREATE VIEW sales.top\_category\_values AS

SELECT DISTINCT SUM (dollar\_value) AS dollar\_value

FROM sales.q5view

GROUP BY category\_id

ORDER BY dollar\_value DESC

LIMIT 20;

CREATE VIEW sales.all\_top\_categories AS

SELECT category\_id

FROM sales.q5view

GROUP BY category\_id

HAVING SUM (dollar\_value) IN (SELECT dollar\_value FROM sales.top\_category\_values);

CREATE VIEW sales.q6\_all 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 (sales.all\_top\_customers cu CROSS JOIN sales.all\_top\_categories ca) LEFT JOIN sales.q4view q

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

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

3) Test query time:

select \* from sales.q6\_all

Query time: 245 MSEC

**Second Scenario:** same as above but this time create indices on materialized views

1. Create indices on underlying views and refresh them

create index q1\_dollar\_value on sales.q1view (dollar\_value)

create index q1\_cust\_id on sales.q1view (customer\_id)

create index q5\_dollar\_value on sales.q5view (dollar\_value)

create index q5\_cat\_id on sales.q5view (category\_id)

refresh materialized view sales.q1view

refresh materialized view sales.q4view

refresh materialized view sales.q5view

2) Test query time:

select \* from sales.q6\_all

Query time: 250 MSEC

**Third Scenario**: create materialized views on higher level views only

1) Create materialized views (notice sales.q6\_all is still not materialized)

drop view sales.top\_customer\_values

drop view sales.top\_category\_values

drop view sales.all\_top\_customers

drop view sales.all\_top\_categories

drop view sales.q6\_all

CREATE materialized VIEW sales.top\_customer\_values AS

SELECT DISTINCT dollar\_value

FROM sales.q1

ORDER BY dollar\_value DESC

LIMIT 20;

CREATE materialized VIEW sales.all\_top\_customers AS

SELECT customer\_id

FROM sales.q1

WHERE dollar\_value IN (SELECT dollar\_value FROM sales.top\_customer\_values);

CREATE materialized VIEW sales.top\_category\_values AS

SELECT DISTINCT SUM (dollar\_value) AS dollar\_value

FROM sales.q5

GROUP BY category\_id

ORDER BY dollar\_value DESC

LIMIT 20;

CREATE materialized VIEW sales.all\_top\_categories AS

SELECT category\_id

FROM sales.q5

GROUP BY category\_id

HAVING SUM (dollar\_value) IN (SELECT dollar\_value FROM sales.top\_category\_values);

refresh materialized view sales.top\_customer\_values

refresh materialized view sales.all\_top\_customers

refresh materialized view sales.top\_category\_values

refresh materialized view sales.all\_top\_categories

CREATE VIEW sales.q6\_all 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 (sales.all\_top\_customers cu CROSS JOIN sales.all\_top\_categories ca) LEFT JOIN sales.q4 q

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

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

2) Test query time:

select \* from sales.q6\_all

Query time: 456 MSEC

**Fourth Scenario**: create materialized views with indices replacing views only where aggregates are being calculated

In this scenario, materialized views are needed on view top\_category\_values, q1, q4, and q5 only. Because I have already created these materialized views I will just drop and recreate virtual views for the remaining views:

1. Create the needed virtual views for the scenario:

drop materialized view sales.top\_customer\_values

drop materialized view sales.all\_top\_customers

drop materialized view sales.all\_top\_categories

drop view sales.q6\_all

CREATE VIEW sales.top\_customer\_values AS

SELECT DISTINCT dollar\_value

FROM sales.q1view

ORDER BY dollar\_value DESC

LIMIT 20;

CREATE VIEW sales.all\_top\_customers AS

SELECT customer\_id

FROM sales.q1view

WHERE dollar\_value IN (SELECT dollar\_value FROM sales.top\_customer\_values);

CREATE VIEW sales.all\_top\_categories AS

SELECT category\_id

FROM sales.q5view

GROUP BY category\_id

HAVING SUM (dollar\_value) IN (SELECT dollar\_value FROM sales.top\_category\_values);

CREATE VIEW sales.q6\_all 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 (sales.all\_top\_customers cu CROSS JOIN sales.all\_top\_categories ca) LEFT JOIN sales.q4 q

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

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

discard all

create index top\_cat\_index on sales.top\_category\_values (dollar\_value)

2) Test query time:

select \* from sales.q6\_all

Query time: 359 MSEC

**Fifth Scenario**: just build materialized view for q6 and create an index on all its columns

1. Drop and recreate q6\_all:

drop view sales.q6\_all

CREATE materialized VIEW sales.q6\_all 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 (sales.all\_top\_customers cu CROSS JOIN sales.all\_top\_categories ca) LEFT JOIN sales.q4 q

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

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

discard all

create index q6all\_index on sales.q6\_all (category\_id, customer\_id, quantity\_sold, dollar\_value)

2) Test query time

select \* from sales.q6\_all

Query time: 181 MSEC

**Conclusion**: the last option above seems to be the most performing from a query time perspective.

CATS CUBE:

Learning from the above, it seems like the best option might be to just create a materialized view on the final query and potentially create an index on it so.

CREATE materialized VIEW cats.weightedMykindLikesMAT (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, cats.inner\_product i, cats.likes l

where u.user\_id = i.x and

l.user\_id = i.y;

create index indx2 on cats.weightedMykindLikesMAT(uid, vid)

Query time: 692 MSEC