Raymond Elward

CSC 553 Project

7 March 2012

Benchmarks from running all 13 queries at once at start:

Q1.1)

est: 71858.890625

run: 141.996645

Q1.2)

est: 72504.585938

run: 136.428536

Q1.3)

est: 72542.632812

run: 142.439553

Q2.1)

est: 73347.390625

run: 69.651182

Q2.2)

est: 73313.382812

run: 21.761561

Q2.3)

est: 73290.820312

run: 23.206359

Q3.1)

est: 71121.437500

run: 92.125390

Q3.2)

est: 70866.632812

run: 131.932380

Q3.3)

est: 70829.929688

run: 143.492966

Q3.4)

est: 70829.257812

run: 141.741235

Q4.1)

est: 80069.164062

run: 17.203689

Q4.2)

est: 76760.718750

run: 115.194987

Q4.3)

est: 73308.757812

run: 146.031711

PART A:

---------------------------------------------------------------------------------------------------------------------

1) I created "CREATE INDEX Q22 ON LINEORDER (lo\_partkey, lo\_suppkey, lo\_orderdate) " index. I choose it because in the query plan on with no indexes it hash joins linorder with the part table. And when I tried to get it to join the supply table first it wouldn't use the index with lo\_suppkey as the first indexed column. It worked pretty well as far as the estimated run times go. The wall clock time is a little strange on our benchmarks for 2.2.

Estimation with index: 18449,

Wall clock with index: 45.48.

2) I created "CREATE INDEX Q31 ON LINEORDER (lo\_custkey, lo\_suppkey, lo\_orderdate, lo\_revenue) " I choose this because it was joining the customer and line order table at the lowest level and I needed to improve the access on the line order table. Also I wanted it to be a covering index so it could be used instead of even needing to go to the table for this query. No noticable improvement. The query plan is not smart enough to use the index. So to that end I used the runstats command to refresh the lineorder table and see if it would use the index in that case. Even though I have a covering index here that is using using the first foriegn join key as its initial indexing column The query optimizer does not use it.

3) I created "CREATE INDEX Q43 ON LINEORDER (lo\_suppkey) CLUSTER" I choose this index because these are the columns with the least selectivity being used in line order. As I learned from the last entry these lower joins in the join tree have a hard time being told to use a certain index, even when it has everything it needs. So I used a clustering index on the join key for the first join operation and it sped things up.

Estimation with index: 13278.385742

4) I created "CREATE INDEX Q1\_4 ON LINEORDER (lo\_discount, lo\_quantity, lo\_suppkey, lo\_custkey, lo\_partkey, lo\_orderdate, lo\_revenue, lo\_supplycost, lo\_extendedprice)" as an index covering needs of both flight 1 and flight 4. It was hard to come to a comprimise for what order to place the queries in. It did not work that well. Only Q1.2 and Q1.3 ended up useing the index at all. These flights have no shared columns so after trying over a dozen indexes I can say its pretty difficult to have a single index that will cover them all.

Q1.1:

est: 71858.890625

run:129.850555

Q1.2

est:67638.148438

run:34.935448

Q1.3

est:66784.078125

run:5.692219

Q4.1

est:80069.164062

run:18.054421

Q4.2

est: 76760.718750

run:15.890936

Q4.3

est: 73308.757812

run:20.590073

5) I will approach the single index for all 13 by finding the most common column with the least selectivity and indexing on that and hopfully the query optimizer will choose it in most cases. I choose "CREATE INDEX QALL ON LINEORDER (lo\_suppkey)" because it is the most common join key for the first join in operations. The query optimizer is not smart enough to use the index.

PART B:

---------------------------------------------------------------------------------------------------------------------

1) The query plan for 4.2 joins lineorder and supplier then dwdate then cust then part. I think we can do better. since its only for this one query i'm going to prejoin everything with

"CREATE TABLE MV42 AS (SELECT d\_year, s\_nation, p\_category, lo\_revenue, lo\_supplycost from lineorder, supplier, dwdate, part, customer where lo\_custkey = c\_custkey and lo\_suppkey = s\_suppkey and lo\_partkey = p\_partkey and lo\_orderdate = d\_datekey and c\_region = 'AMERICA' and s\_region = 'AMERICA' and (d\_year = 1997 or d\_year = 1998) and (p\_mfgr = 'MFGR#1' or p\_mfgr = 'MFGR#2' ) ) DATA INITIALLY DEFERRED REFRESH DEFERRED"

and issue the query

"select d\_year, s\_nation, p\_category, sum(CAST (lo\_revenue - lo\_supplycost as BIGINT)) as profit from MV42 group by d\_year, s\_nation, p\_category order by d\_year, s\_nation, p\_category".

And the improvement was:

Q4.2:

est:171.275604

run: 1.330858

2) For the query plan in flight 4 I want to pre compute all the common joins. I created the materialized view:

" CREATE TABLE MV4 AS (SELECT d\_year, s\_nation, p\_category, lo\_revenue, lo\_supplycost, s\_region, s\_city, c\_nation, c\_region, p\_brand1, p\_mfgr from lineorder, supplier, dwdate, part, customer where lo\_custkey = c\_custkey and lo\_suppkey = s\_suppkey and lo\_orderdate = d\_datekey and lo\_partkey = p\_partkey) DATA INITIALLY DEFERRED REFRESH DEFERRED"

I changed the queries to: "

select d\_year, c\_nation, sum(CAST(lo\_revenue - lo\_supplycost as BIGINT)) as profit from MV4 where c\_region = 'AMERICA' and s\_region = 'AMERICA' and (p\_mfgr = 'MFGR#1' or p\_mfgr = 'MFGR#2') group by d\_year, c\_nation, p\_category order by d\_year, c\_nation, p\_category

select d\_year, s\_nation, p\_category, sum(CAST(lo\_revenue - lo\_supplycost as BIGINT)) as profit from MV4 where c\_region = 'AMERICA' and s\_region = 'AMERICA' and (d\_year = 1997 or d\_year = 1998) and (p\_mfgr = 'MFGR#1' or p\_mfgr = 'MFGR#2') group by d\_year, s\_nation, p\_category order by d\_year, s\_nation, p\_category

select d\_year, s\_city p\_brand1, sum(CAST(lo\_revenue - lo\_supplycost as BIGINT)) as profit from MV4 where s\_nation = 'UNITED STATES' and (d\_year = 1997 or d\_year = 1998) and p\_category = 'MFGR#14' group by d\_year, s\_city, p\_brand1 order by d\_year, s\_city, p\_brand1

"

And the improvements were:

Q4.1:95319.843750

est:22.319338

run: 23.808671

Q4.2:

est:95380.187500

run:1.422037

Q4.3:

est:95319.718750

run:21.765912

interesting here that runtimes are a lot quicker than our baseline, but the estimations gived are a little bit worse for the most part.

3) 1.3 and 4.2 don't have much in common… but I made a materialized view that combines dwdate and lineorder so 1.3 will have full benifit while 4.2 will have a small benifit but reordering the join so dwdate and lineorder are combined first.

"

CREATE TABLE MV13\_42 AS (SELECT lo\_extendedprice, lo\_discount, lo\_revenue, lo\_supplycost, lo\_suppkey, lo\_partkey, lo\_custkey, d\_weeknuminyear, d\_year, lo\_quantity from lineorder, dwdate where lo\_orderdate = d\_datekey) DATA INITIALLY DEFERRED REFRESH DEFERRED

"

The queries became: "

select sum(CAST(lo\_extendedprice \* lo\_discount as BIGINT)) as revenue from MV13\_42 where d\_weeknuminyear = 6 and d\_year = 1994 and lo\_discount between 5 and 7 and lo\_quantity between 26 and 35

select d\_year s\_nation, p\_category, sum(CAST (lo\_revenue - lo\_supplycost as BIGINT)) as profit from MV13\_42, customer, supplier, part where lo\_custkey = c\_custkey and lo\_suppkey = s\_suppkey and lo\_partkey = p\_partkey and c\_region = 'AMERICA' and s\_region = 'AMERICA' and (d\_year = 1997 or d\_year = 1998) and (p\_mfgr = 'MFGR#1' or p\_mfgr = 'MFGR#2') group by d\_year, s\_nation, p\_category order by d\_year, s\_nation, p\_category

"

And the improvements were:

Q1.2:

est: 41212.117188

run: 10.249072

Q4.2:

est: 36983.449219

run: 74.743759

Query from flight 1 got a much better improvement.

4) Flight 2 MV I picked out what they all had in common and put it in the view.

"

CREATE TABLE MV2 AS (SELECT sum(lo\_revenue) as revenue, d\_year, p\_brand1, s\_region, p\_category from lineorder, dwdate, part, supplier where lo\_orderdate = d\_datekey and lo\_partkey = p\_partkey and lo\_suppkey = s\_suppkey group by d\_year, p\_brand1, s\_region, p\_category) DATA INITIALLY DEFERRED REFRESH DEFERRED

"

The new queries will be:

"

select revenue, d\_year, p\_brand1 from MV2 where p\_category = 'MFGR#12' and s\_region = 'AMERICA' order by d\_year, p\_brand1

select revenue, d\_year, p\_brand1 from MV2 where p\_brand1 between 'MFGR#2221' and 'MFGR#2228' and s\_region = 'ASIA' order by d\_year, p\_brand1

select revenue, d\_year, p\_brand1 from MV2 where p\_brand1 = 'MFGR#2239' and s\_region = 'EUROPE' order by d\_year, p\_brand1

"

and the improvements were:

Q2.1:

est: 267.501892

run: 1.490543

Q2.2:

est: 267.834259

run: 1.399686

Q2.3

est: 267.493042

run: 1.635781

5) Flight 4 MV I picked out what they all had in common and put it in the view.

""""

CREATE TABLE MV4\_PRE AS (select d\_year, s\_nation, p\_category, sum(CAST(lo\_revenue - lo\_supplycost as BIGINT)) as profit, c\_nation, c\_region, s\_region, p\_mfgr, c\_city, s\_city, p\_brand1 from lineorder, dwdate, customer, supplier, part where lo\_custkey = c\_custkey and lo\_suppkey = s\_suppkey and lo\_partkey = p\_partkey and lo\_orderdate = d\_datekey group by d\_year, c\_nation, s\_nation, p\_category, s\_city, p\_brand1, c\_city, p\_mfgr, s\_region, c\_region ) DATA INITIALLY DEFERRED REFRESH DEFERRED

refresh table DB2INST1.MV4\_PRE

"""

And the queries to check on this new MV is:

"""

-- Q4.1

select d\_year, c\_nation, sum(profit) from MV4\_PRE where c\_region = 'AMERICA' and s\_region = 'AMERICA' and (p\_mfgr = 'MFGR#1' or p\_mfgr = 'MFGR#2') group by d\_year, c\_nation order by d\_year, c\_nation

-- Q4.2

select d\_year, s\_nation, p\_category, sum(profit) from MV4\_PRE where c\_region = 'AMERICA' and s\_region = 'AMERICA' and (d\_year = 1997 or d\_year = 1998) and (p\_mfgr = 'MFGR#1' or p\_mfgr = 'MFGR#2') group by d\_year, s\_nation, p\_category order by d\_year, s\_nation, p\_category

-- Q4.3

select d\_year, s\_city, p\_brand1, sum(profit) from MV4\_PRE where s\_nation = 'UNITED STATES' and (d\_year = 1997 or d\_year = 1998) and p\_category = 'MFGR#14' group by d\_year, s\_city, p\_brand1 order by d\_year, s\_city, p\_brand1

"""

The queries resulted in these run times It evens them all out and saves a lot of time on 4.2 and 4.3:

Q41)

est: 98593.453125

run: 25.562376

Q42)

est: 98662.750000

run: 26.612480

Q43)

est: 98457.976562

run: 26.106594

6) The only join in common with all queries is the dwdate & lineorder join. So I will make a MV that combines those 2 that all queries can use to optimize a bit.

"

CREATE TABLE MVALL AS (SELECT \* FROM dwdate, lineorder where lo\_orderdate = d\_datekey) DATA INITIALLY DEFERRED REFRESH DEFERRED

"

I rewrite all the queries to:

"

-- Q1.1

select sum(CAST(lo\_extendedprice\*lo\_discount as BIGINT)) as revenue from MVALL where d\_year = 1993 and lo\_discount between 1 and 3 and lo\_quantity < 25

-- Q1.2

select sum(CAST(lo\_extendedprice\*lo\_discount as BIGINT)) as revenue from MVALL where d\_yearmonthnum = 199401 and lo\_discount between 4 and 6 and lo\_quantity between 26 and 35

-- Q1.3

select sum(CAST(lo\_extendedprice\*lo\_discount as BIGINT)) as revenue from MVALL where d\_weeknuminyear = 6 and d\_year = 1994 and lo\_discount between 5 and 7 and lo\_quantity between 26 and 35

-- Q2.1

select sum(lo\_revenue), d\_year, p\_brand1 from MVALL, part, supplier where lo\_partkey = p\_partkey and lo\_suppkey = s\_suppkey and p\_category = 'MFGR#12' and s\_region = 'AMERICA' group by d\_year, p\_brand1 order by d\_year, p\_brand1

-- Q2.2

select sum(lo\_revenue), d\_year, p\_brand1 from MVALL, part, supplier where lo\_partkey = p\_partkey and lo\_suppkey = s\_suppkey and p\_brand1 between 'MFGR#2221' and 'MFGR#2228' and s\_region = 'ASIA' group by d\_year, p\_brand1 order by d\_year, p\_brand1

-- Q2.3

select sum(lo\_revenue), d\_year, p\_brand1 from MVALL, part, supplier where lo\_partkey = p\_partkey and lo\_suppkey = s\_suppkey and p\_brand1= 'MFGR#2239' and s\_region = 'EUROPE' group by d\_year, p\_brand1 order by d\_year, p\_brand1

-- Q3.1

select c\_nation, s\_nation, d\_year, sum(CAST (lo\_revenue as BIGINT)) as revenue from customer, MVALL, supplier where lo\_custkey = c\_custkey and lo\_suppkey = s\_suppkey and c\_region = 'ASIA' and s\_region = 'ASIA' and d\_year >= 1992 and d\_year <= 1997 group by c\_nation, s\_nation, d\_year order by d\_year asc, revenue desc

-- Q3.2

select c\_city, s\_city, d\_year, sum(CAST (lo\_revenue as BIGINT)) as revenue from customer, MVALL, supplier where lo\_custkey = c\_custkey and lo\_suppkey = s\_suppkey and c\_nation = 'UNITED STATES' and s\_nation = 'UNITED STATES' and d\_year >= 1992 and d\_year <= 1997 group by c\_city, s\_city, d\_year order by d\_year asc, revenue desc

-- Q3.3

select c\_city, s\_city, d\_year, sum(CAST (lo\_revenue as BIGINT)) as revenue from customer, MVALL, supplier where lo\_custkey = c\_custkey and lo\_suppkey = s\_suppkey and (c\_city='UNITED KI1' or c\_city='UNITED KI5') and (s\_city='UNITED KI1' or s\_city='UNITED KI5') and d\_year >= 1992 and d\_year <= 1997 group by c\_city, s\_city, d\_year order by d\_year asc, revenue desc

-- Q3.4

select c\_city, s\_city, d\_year, sum(CAST (lo\_revenue as BIGINT)) as revenue from customer, MVALL, supplier where lo\_custkey = c\_custkey and lo\_suppkey = s\_suppkey and (c\_city='UNITED KI1' or c\_city='UNITED KI5') and (s\_city='UNITED KI1' or s\_city='UNITED KI5') and d\_yearmonth = 'Dec1997' group by c\_city, s\_city, d\_year order by d\_year asc, revenue desc

-- Q4.1

select d\_year, c\_nation, sum(CAST (lo\_revenue - lo\_supplycost as BIGINT)) as profit from customer, supplier, part, MVALL where lo\_custkey = c\_custkey and lo\_suppkey = s\_suppkey and lo\_partkey = p\_partkey and c\_region = 'AMERICA' and s\_region = 'AMERICA' and (p\_mfgr = 'MFGR#1' or p\_mfgr = 'MFGR#2') group by d\_year, c\_nation order by d\_year, c\_nation

-- Q4.2

select d\_year, s\_nation, p\_category, sum(CAST (lo\_revenue - lo\_supplycost as BIGINT)) as profit from customer, supplier, part, MVALL where lo\_custkey = c\_custkey and lo\_suppkey = s\_suppkey and lo\_partkey = p\_partkey and c\_region = 'AMERICA' and s\_region = 'AMERICA' and (d\_year = 1997 or d\_year = 1998) and (p\_mfgr = 'MFGR#1' or p\_mfgr = 'MFGR#2') group by d\_year, s\_nation, p\_category order by d\_year, s\_nation, p\_category

-- Q4.3

select d\_year, s\_city, p\_brand1, sum(CAST (lo\_revenue - lo\_supplycost as BIGINT)) as profit from customer, supplier, part, MVALL where lo\_custkey = c\_custkey and lo\_suppkey = s\_suppkey and lo\_partkey = p\_partkey and s\_nation = 'UNITED STATES' and (d\_year = 1997 or d\_year = 1998) and p\_category = 'MFGR#14' group by d\_year, s\_city, p\_brand1 order by d\_year, s\_city, p\_brand1

"

Q1.1)

est: 159643.250000

run: 43.490700

Q1.2)

est: 159687.375000

run: 37.493477

Q1.3)

est: 159698.140625

run: 35.279464

Q2.1)

est: 160359.781250

run: 41.454453

Q2.2)

est: 161414.468750

run: 40.377466

Q2.3)

est: 160356.171875

run: 41.260344

Q3.1)

est: 159759.281250

run: 38.472634

Q3.2)

est: 159759.500000

run: 40.177137

Q3.3)

est: 159660.125000

run: 41.066288

Q3.4)

est: 158422.203125

run: 36.030667

Q4.1)

est: 159099.640625

run: 42.885501

Q4.2)

est: 157493.890625

run: 109.805740

Q4.3)

est: 157486.328125

run: 61.349951

PART C:

---------------------------------------------------------------------------------------------------------------------

1) Structures created:

"

CREATE TABLE MV2\_ALL AS (SELECT sum(lo\_revenue) as total\_sum, d\_year, p\_brand1, s\_region, p\_category from lineorder, part, supplier, dwdate where lo\_orderdate = d\_datekey and lo\_suppkey = s\_suppkey and lo\_partkey = p\_partkey group by d\_year, p\_brand1, s\_region, p\_category) DATA INITIALLY DEFERRED REFRESH DEFERRED

CREATE INDEX MV2\_ALL\_INDEX ON MV2\_ALL (d\_year, p\_brand1) CLUTSER

"

and I changed the queries to run under this as:

"

select total\_sum, d\_year, p\_brand1 from MV2\_ALL where p\_category = 'MFGR#12' and s\_region = 'AMERICA' order by d\_year, p\_brand1

select total\_sum, d\_year, p\_brand1 from MV2\_ALL where p\_brand1 between 'MFGR#2221' and 'MFGR#2228' and s\_region = 'ASIA' order by d\_year, p\_brand1

select total\_sum, d\_year, p\_brand1 from MV2\_ALL where p\_brand1= 'MFGR#2239' and s\_region = 'EUROPE' order by d\_year, p\_brand1

"

This gave performance of:

Q2.1)

est: 267.501892

run: 1.498023

Q2.2)

est: 267.834259

run: 1.512223

Q2.3)

est: 267.493042

run: 1.470636

2) Structures created, I used the where clause from the date because all of the queries ask from things between those dates.:

"

CREATE TABLE MV3\_ALL AS (SELECT c\_nation, s\_nation, d\_year, lo\_revenue, c\_region, s\_region, c\_city, s\_city, d\_yearmonth FROM customer, lineorder, supplier, dwdate WHERE lo\_custkey = c\_custkey and lo\_suppkey = s\_suppkey and lo\_orderdate = d\_datekey and d\_year >= 1992 and d\_year <= 1997) DATA INITIALLY DEFERRED REFRESH DEFERRED

CREATE INDEX MV3\_ALL\_INDEX ON MV3\_ALL (d\_year, c\_region, c\_nation, c\_city) CLUSTER

"

And then I change the queries to run as:

"

-- Q3.1

select c\_nation, s\_nation, d\_year, sum(CAST (lo\_revenue as BIGINT)) as revenue from MV3\_ALL where c\_region = 'ASIA' and s\_region = 'ASIA' group by c\_nation, s\_nation, d\_year order by d\_year asc, revenue desc

-- Q3.2

select c\_city, s\_city, d\_year, sum(CAST (lo\_revenue as BIGINT)) as revenue from MV3\_ALL where c\_nation = 'UNITED STATES' and s\_nation = 'UNITED STATES' group by c\_city, s\_city, d\_year order by d\_year asc, revenue desc

-- Q3.3

select c\_city, s\_city, d\_year, sum(CAST (lo\_revenue as BIGINT)) as revenue from MV3\_ALL where (c\_city='UNITED KI1' or c\_city='UNITED KI5') and (s\_city='UNITED KI1' or s\_city='UNITED KI5') group by c\_city, s\_city, d\_year order by d\_year asc, revenue desc

-- Q3.4

select c\_city, s\_city, d\_year, sum(CAST (lo\_revenue as BIGINT)) as revenue from MV3\_ALL where (c\_city='UNITED KI1' or c\_city='UNITED KI5') and (s\_city='UNITED KI1' or s\_city='UNITED KI5') and d\_yearmonth = 'Dec1997' group by c\_city, s\_city, d\_year order by d\_year asc, revenue desc

"

This gave performance of:

Q3.1)

est: 70028.078125

run: 16.225848

Q3.2)

est: 70028.078125

run: 15.588936

Q3.3)

est: 67998.046875

run: 13.418569

Q3.4)

est: 78602.882812

run: 18.455027

3)

I created the MV and clustering index as:

"""

CREATE TABLE MV4\_C3 AS (select d\_year, s\_nation, p\_category, lo\_revenue, lo\_supplycost, c\_nation, c\_region, s\_region, p\_mfgr, c\_city, s\_city, p\_brand1 from lineorder, dwdate, customer, supplier, part where lo\_custkey = c\_custkey and lo\_suppkey = s\_suppkey and lo\_partkey = p\_partkey and lo\_orderdate = d\_datekey) DATA INITIALLY DEFERRED REFRESH DEFERRED

refresh table DB2INST1. MV4\_C3

CREATE INDEX MV4\_C3\_INDEX ON MV4\_C3 (d\_year) CLUTSER

"""

I chose to cluster index on d\_year, because its what is commonly grouping all three queries.

I used the queries here to test speeds:

"""

-- Q4.1

select d\_year, c\_nation, sum(CAST (lo\_revenue - lo\_supplycost as BIGINT)) as profit from MV4\_C3 where c\_region = 'AMERICA' and s\_region = 'AMERICA' and (p\_mfgr = 'MFGR#1' or p\_mfgr = 'MFGR#2') group by d\_year, c\_nation order by d\_year, c\_nation

-- Q4.2

select d\_year, s\_nation, p\_category, sum(CAST (lo\_revenue - lo\_supplycost as BIGINT)) as profit from MV4\_C3 where c\_region = 'AMERICA' and s\_region = 'AMERICA' and (d\_year = 1997 or d\_year = 1998) and (p\_mfgr = 'MFGR#1' or p\_mfgr = 'MFGR#2') group by d\_year, s\_nation, p\_category order by d\_year, s\_nation, p\_category

-- Q4.3

select d\_year, s\_city, p\_brand1, sum(CAST (lo\_revenue - lo\_supplycost as BIGINT)) as profit from MV4\_C3 where s\_nation = 'UNITED STATES' and (d\_year = 1997 or d\_year = 1998) and p\_category = 'MFGR#14' group by d\_year, s\_city, p\_brand1 order by d\_year, s\_city, p\_brand1

"""

The queries resulted in these run times, which was a pretty go improvement:

Q41)

est:106633.750000

run:26.723250

Q42)

est:45.405293

run: 43.102628

Q43)

est:45.405289

run: 43.293392

4) For all 13 queries I created the MV + 3 indexes. I picked to cluster index on the supply key because it connects this view to the supply table which is where it is joined most often. then I index on the year because that is called in nearly every query. Then I have an index on the cust key so that the joins with the cust table move quicker :

"

CREATE TABLE MVALL AS (SELECT \* FROM dwdate, lineorder where lo\_orderdate = d\_datekey) DATA INITIALLY DEFERRED REFRESH DEFERRED

CREATE INDEX MVALL\_INDEX1 ON MVALL (lo\_suppkey) CLUSTER

CREATE INDEX MVALL\_INDEX2 ON MVALL (d\_year)

CREATE INDEX MVALL\_INDEX3 ON MVALL (lo\_custkey)

"

I used the same queries from PART B 6) and came up with these results. Since the query optimizer wasn't picking the indexes in some instances we came out with some rather crazy long wait times for joins(see all of flight 2 for example). I believe using the date table for the first join is a terrible idea for flight two, but comprimises had to be made to give little speed boosts to everywhere else (I literally had to leave this one running overnight because it took so long).:

Q1.1)

est: 36926.523438

run: 72.596267

Q1.2)

est: 159687.375000

run: 36.969160

Q1.3)

est: 36932.875000

run: 71.828710

Q2.1)

est: 28012.591797

run: 1213.391967

Q2.2)

est: 29067.281250

run: 1426.665990

Q2.3)

est: 28008.974609

run: 1227.328023

Q3.1)

est: 25575.919922

run: 1453.348208

Q3.2)

est: 25576.142578

run: 249.517941

Q3.3)

est: 854.649658

run: 2.857807

Q3.4)

est: 854.649597

run: 2.358716

Q4.1)

est: 26752.431641

run: 1227.295249

Q4.2)

est: 105.947128

run: 137.287658

Q4.3)

est: 98.383316

run: 77.176903

PART D::

---------------------------------------------------------------------------------------------------------------------

1)500MB with just indexes: (in email you told me I didn't need exact sizes of indexes because db2 was not being precise about index lists / sizes anyway. I'm taking estimations from the queries given in assignment 5.)

With 500 MB of space and only doing indexes.

CREATE INDEX D1\_1 ON LINEORDER (lo\_datekey)

CREATE INDEX D1\_2 ON LINEORDER (lo\_suppkey)

CREATE INDEX D1\_3 ON LINEORDER (lo\_custkey)

CREATE INDEX D1\_4 ON LINEORDER (lo\_partkey)

With a cost of about 100mb for each index these give us the best bang for our buck when doing joins. The indexes here will work for any kind of join on the lineorder table because it uses all the foreign keys in the schema

2) 1000MB with just indexes:

Entire query index plus a couple for the most time consuming queries

Index on flight 1 and flight 3 because they are the most time consuming.

Here we have enough space to get specific on some flights. We can still do the all 4 foreign key indexes from part 1, but now we have a chance to optimize some flights individually.

so about 400 from those indexes plus:

CREATE INDEX D2\_1 ON LINEORDER (lo\_discount, lo\_quantity, lo\_orderdate, lo\_extendedprice)

will be good for the first flight and will give us a good improvements est cost about 174mb

CREATE INDEX D2\_2 ON LINEORDER (lo\_suppkey, lo\_partkey, lo\_revenue, lo\_orderdate)

This is a covering index for the second flight. It will drastically improve the second flight when used and is about the size of 178mb.

CREATE INDEX D2\_3 ON LINEORDER (lo\_suppkey, lo\_custkey, lo\_revenue, lo\_orderdate)

This is a covering index for the third flight. It will drastically improve the second flight when used and is about the size of 174mb.

CREATE INDEX D2\_4 ON LINEORDER (lo\_suppkey, lo\_custkey, lo\_revenue, lo\_orderdate, lo\_supplycost)

This is a covering index for the third flight. It will drastically improve the second flight when used and is about the size of 200mb.

All these indexes will give great improvements on the executions of the queries if used properly by the query optimizer. The problem is that it would be too big for the only 1000mb… so I would take off D2\_2 and that would get it around 950mb. I would do this because flight 2 is benchmarked as the quickest flight, so letting the others catch up first would work best.

3) 1500MB with just indexes:

Index on each flight individually.

With this much space we can get into the details of the each query and optimize each one by its specific predicates and optimize each on lineorder. since line order indexes with four indexed columns costs.

4) 600 MB with index/MV:

I wondered if this was a trick question because by making specific MVs for each query it takes up the least amount of space out of anything and gets the best possible runtime, because all that needs to be done when querying the system is read the new materialized view. I used the MVs by loading the following script:

"""

-- Q1.1

CREATE TABLE MV11\_FULL AS (select sum(CAST(lo\_extendedprice\*lo\_discount as BIGINT)) as revenue from lineorder, dwdate where lo\_orderdate = d\_datekey and d\_year = 1993 and lo\_discount between 1 and 3 and lo\_quantity < 25) DATA INITIALLY DEFERRED REFRESH DEFERRED

refresh table DB2INST1.MV11\_FULL

-- Q1.2

CREATE TABLE MV12\_FULL AS (select sum(CAST(lo\_extendedprice\*lo\_discount as BIGINT)) as revenue from lineorder, dwdate where lo\_orderdate = d\_datekey and d\_yearmonthnum = 199401 and lo\_discount between 4 and 6 and lo\_quantity between 26 and 35) DATA INITIALLY DEFERRED REFRESH DEFERRED

refresh table DB2INST1.MV12\_FULL

-- Q1.3

CREATE TABLE MV13\_FULL AS (select sum(CAST(lo\_extendedprice\*lo\_discount as BIGINT)) as revenue from lineorder, dwdate where lo\_orderdate = d\_datekey and d\_weeknuminyear = 6 and d\_year = 1994 and lo\_discount between 5 and 7 and lo\_quantity between 26 and 35) DATA INITIALLY DEFERRED REFRESH DEFERRED

refresh table DB2INST1.MV13\_FULL

-- Q2.1

CREATE TABLE MV21\_FULL AS (select sum(lo\_revenue) as sum\_revenue, d\_year, p\_brand1 from lineorder, dwdate, part, supplier where lo\_orderdate = d\_datekey and lo\_partkey = p\_partkey and lo\_suppkey = s\_suppkey and p\_category = 'MFGR#12' and s\_region = 'AMERICA' group by d\_year, p\_brand1) DATA INITIALLY DEFERRED REFRESH DEFERRED

refresh table DB2INST1.MV21\_FULL

-- Q2.2

CREATE TABLE MV22\_FULL AS (select sum(lo\_revenue) as sum\_revenue, d\_year, p\_brand1 from lineorder, dwdate, part, supplier where lo\_orderdate = d\_datekey and lo\_partkey = p\_partkey and lo\_suppkey = s\_suppkey and p\_brand1 between 'MFGR#2221' and 'MFGR#2228' and s\_region = 'ASIA' group by d\_year, p\_brand1) DATA INITIALLY DEFERRED REFRESH DEFERRED

refresh table DB2INST1.MV22\_FULL

-- Q2.3

CREATE TABLE MV23\_FULL AS (select sum(lo\_revenue) as sum\_revenue, d\_year, p\_brand1 from lineorder, dwdate, part, supplier where lo\_orderdate = d\_datekey and lo\_partkey = p\_partkey and lo\_suppkey = s\_suppkey and p\_brand1= 'MFGR#2239' and s\_region = 'EUROPE' group by d\_year, p\_brand1) DATA INITIALLY DEFERRED REFRESH DEFERRED

refresh table DB2INST1.MV23\_FULL

-- Q3.1

CREATE TABLE MV31\_FULL AS (select c\_nation, s\_nation, d\_year, sum(CAST (lo\_revenue as BIGINT)) as revenue from customer, lineorder, supplier, dwdate where lo\_custkey = c\_custkey and lo\_suppkey = s\_suppkey and lo\_orderdate = d\_datekey and c\_region = 'ASIA' and s\_region = 'ASIA' and d\_year >= 1992 and d\_year <= 1997 group by c\_nation, s\_nation, d\_year ) DATA INITIALLY DEFERRED REFRESH DEFERRED

refresh table DB2INST1.MV31\_FULL

-- Q3.2

CREATE TABLE MV32\_FULL AS (select c\_city, s\_city, d\_year, sum(CAST (lo\_revenue as BIGINT)) as revenue from customer, lineorder, supplier, dwdate where lo\_custkey = c\_custkey and lo\_suppkey = s\_suppkey and lo\_orderdate = d\_datekey and c\_nation = 'UNITED STATES' and s\_nation = 'UNITED STATES' and d\_year >= 1992 and d\_year <= 1997 group by c\_city, s\_city, d\_year ) DATA INITIALLY DEFERRED REFRESH DEFERRED

refresh table DB2INST1.MV32\_FULL

-- Q3.3

CREATE TABLE MV33\_FULL AS (select c\_city, s\_city, d\_year, sum(CAST (lo\_revenue as BIGINT)) as revenue from customer, lineorder, supplier, dwdate where lo\_custkey = c\_custkey and lo\_suppkey = s\_suppkey and lo\_orderdate = d\_datekey and (c\_city='UNITED KI1' or c\_city='UNITED KI5') and (s\_city='UNITED KI1' or s\_city='UNITED KI5') and d\_year >= 1992 and d\_year <= 1997 group by c\_city, s\_city, d\_year) DATA INITIALLY DEFERRED REFRESH DEFERRED

refresh table DB2INST1.MV33\_FULL

-- Q3.4

CREATE TABLE MV34\_FULL AS (select c\_city, s\_city, d\_year, sum(CAST (lo\_revenue as BIGINT)) as revenue from customer, lineorder, supplier, dwdate where lo\_custkey = c\_custkey and lo\_suppkey = s\_suppkey and lo\_orderdate = d\_datekey and (c\_city='UNITED KI1' or c\_city='UNITED KI5') and (s\_city='UNITED KI1' or s\_city='UNITED KI5') and d\_yearmonth = 'Dec1997' group by c\_city, s\_city, d\_year) DATA INITIALLY DEFERRED REFRESH DEFERRED

refresh table DB2INST1.MV34\_FULL

-- Q4.1

CREATE TABLE MV41\_FULL AS (select d\_year, c\_nation, sum(CAST (lo\_revenue - lo\_supplycost as BIGINT)) as profit from dwdate, customer, supplier, part, lineorder where lo\_custkey = c\_custkey and lo\_suppkey = s\_suppkey and lo\_partkey = p\_partkey and lo\_orderdate = d\_datekey and c\_region = 'AMERICA' and s\_region = 'AMERICA' and (p\_mfgr = 'MFGR#1' or p\_mfgr = 'MFGR#2') group by d\_year, c\_nation) DATA INITIALLY DEFERRED REFRESH DEFERRED

refresh table DB2INST1.MV41\_FULL

-- Q4.2

CREATE TABLE MV42\_FULL AS (select d\_year, s\_nation, p\_category, sum(CAST (lo\_revenue - lo\_supplycost as BIGINT)) as profit from dwdate, customer, supplier, part, lineorder where lo\_custkey = c\_custkey and lo\_suppkey = s\_suppkey and lo\_partkey = p\_partkey and lo\_orderdate = d\_datekey and c\_region = 'AMERICA' and s\_region = 'AMERICA' and (d\_year = 1997 or d\_year = 1998) and (p\_mfgr = 'MFGR#1' or p\_mfgr = 'MFGR#2') group by d\_year, s\_nation, p\_category) DATA INITIALLY DEFERRED REFRESH DEFERRED

refresh table DB2INST1.MV42\_FULL

-- Q4.3

CREATE TABLE MV43\_FULL AS (select d\_year, s\_city, p\_brand1, sum(CAST (lo\_revenue - lo\_supplycost as BIGINT)) as profit from dwdate, customer, supplier, part, lineorder where lo\_custkey = c\_custkey and lo\_suppkey = s\_suppkey and lo\_partkey = p\_partkey and lo\_orderdate = d\_datekey and s\_nation = 'UNITED STATES' and (d\_year = 1997 or d\_year = 1998) and p\_category = 'MFGR#14' group by d\_year, s\_city, p\_brand1) DATA INITIALLY DEFERRED REFRESH DEFERRED

refresh table DB2INST1.MV43\_FULL

"""

After that I ran the queries to see the run times:

"""

SELECT \* FROM MV11\_FULL

SELECT \* FROM MV12\_FULL

SELECT \* FROM MV13\_FULL

SELECT \* FROM MV21\_FULL ORDER BY d\_year, p\_brand1

SELECT \* FROM MV22\_FULL ORDER BY d\_year, p\_brand1

SELECT \* FROM MV23\_FULL ORDER BY d\_year, p\_brand1

SELECT \* FROM MV31\_FULL ORDER BY d\_year ASC, revenue DESC

SELECT \* FROM MV32\_FULL ORDER BY d\_year ASC, revenue DESC

SELECT \* FROM MV33\_FULL ORDER BY d\_year ASC, revenue DESC

SELECT \* FROM MV34\_FULL ORDER BY d\_year ASC, revenue DESC

SELECT \* FROM MV41\_FULL ORDER BY d\_year, c\_nation

SELECT \* FROM MV42\_FULL ORDER BY d\_year, s\_nation, p\_category

SELECT \* FROM MV43\_FULL ORDER BY d\_year, s\_city, p\_brand1

"""

The run speeds were (do I remember something about more points for best in class, because not a single query goes longer than 1.5 seconds here?)::

Q1.1)

est: 7.572975

run: 1.416335

Q1.2)

est: 7.662697

run: 1.462570

Q1.3)

est: 7.662697

run: 1.407597

Q2.1)

est: 15.505921

run: 1.395982

Q2.2)

est: 7.732398

run: 1.408482

Q2.3)

est: 7.732398

run: 1.459519

Q3.1)

est: 7.734639

run: 1.425059

Q3.2)

est: 31.089220

run: 1.431045

Q3.3)

est: 7.734639

run: 1.409780

Q3.4)

est: 7.734639

run: 1.441002

Q4.1)

est: 7.732398

run: 1.427320

Q4.2)

est: 7.758903

run: 1.437412

Q4.3)

est: 38.497852

run: 1.471478

And then I totaled up the sizes which were:

MV11\_FULL = 0.00 (only 1 row… so less than 0.00mb)

MV12\_FULL = 0.00 (only 1 row… so less than 0.00mb)

MV13\_FULL = 0.00 (only 1 row… so less than 0.00mb)

MV21\_FULL = 0.01 (only 256 rows)

MV22\_FULL = 0.00 (only 56 rows… so less than 0.00mb)

MV23\_FULL = 0.00 (only 7 rows… so less than 0.00mb)

MV31\_FULL = 0.00 (only 150 rows… so less than 0.00mb)

MV32\_FULL = 0.03 (only 596 rows)

MV33\_FULL = 0.00 (only 24 rows… so less than 0.00mb)

MV34\_FULL = 0.00 (only 3 rows… so less than 0.00mb)

MV41\_FULL = 0.00 (only 35 rows… so less than 0.00mb)

MV42\_FULL = 0.00 (only 100 rows… so less than 0.00mb)

MV43\_FULL = 0.03 (only 700 rows)

That is a grand total of 0.07 mb of MVs to get all the queries in less than a second and a half. The question then remains what to do with the remaining 599.93 mb of space. The draw back to this approach is updates and inserts, but since this is a large datastore where updates rarely happen and inserts are batched nightly the refresh of the views only takes a limited time after all the inserts. Another big problem with the fully pre-aggregated MVs is that if a query were to change ever slightly there would be a lot of overhead just to adjust for those changes.

5) 1.2GB with index/MV:

per our discussion I will not be using pre-aggregates to solve this issue. It makes this problem much more difficult. I'm going to check MV sizes on the individual flights. Since my MV3 from C2 is about 500mb it would not be sustainable to build that size MVs for all the flights, but all the flights are different so lets see if we can get smaller ones for others to make up space. The MV123\_FULL from assignment 5 is only 200mb. 500mb left for just flight 2 and 4. The MV I came up with for flight 4

" CREATE TABLE MV4\_FULL AS (select d\_year, s\_nation, p\_category, lo\_revenue, lo\_supplycost, c\_nation, c\_region, s\_region, p\_mfgr, c\_city, s\_city, p\_brand1 from lineorder, dwdate, customer, supplier, part where lo\_custkey = c\_custkey and lo\_suppkey = s\_suppkey and lo\_partkey = p\_partkey and lo\_orderdate = d\_datekey) DATA INITIALLY DEFERRED REFRESH DEFERRED"

was 700mb. Not enough space for that one. We'll retry it with only pre-joining 2 tables instead of 4.

" CREATE TABLE MV4\_FULL2 AS (select s\_nation, lo\_revenue, lo\_supplycost, lo\_orderdate, lo\_partkey, c\_nation, c\_region, s\_region, c\_city, s\_city from lineorder, customer, supplier where lo\_custkey = c\_custkey and lo\_suppkey = s\_suppkey) DATA INITIALLY DEFERRED REFRESH DEFERRED"

Now it only costs 500mb. Lets see what we can get out of flight 2. And we may have to do the same thing we did here for flight 3 (reducing the number of joins). With a full join for flight 2 we have :

"

CREATE TABLE MV2\_FULL AS (select lo\_revenue, d\_year, p\_brand1, p\_category, s\_region from lineorder, dwdate, part, supplier where lo\_partkey = p\_partkey and lo\_suppkey = s\_suppkey and lo\_orderdate = d\_datekey) DATA INITIALLY DEFERRED REFRESH DEFERRED

"

which costs 300mb. Lets see if we can lower the cost on flight 3. Reducing the MV on flight 3 to only 1 join we get:

"CREATE TABLE MV3\_FULL2 AS (select lo\_orderdate, lo\_revenue, lo\_custkey, s\_nation, s\_region, s\_city from lineorder, supplier where lo\_suppkey = s\_suppkey) DATA INITIALLY DEFERRED REFRESH DEFERRED"

which costs only 300mb. So the plan for only 1500mb of space is to use MV2\_FULL(300mb), MV3\_FULL2(300mb), MV123\_FULL(200mb), and MV4\_FULL2(600mb).

6) 1.8GB with index/MV:

With 1.8gb we have enough to make a MV that is a prejoin of all the tables that costs about 1200mb. And now we just add indexes to it for each individual flight. to call quickly to the one giant MV containing everything. With this one giant MV we can create individual indexes for each flight to optimize the way the flights are accessed.

(took 3 different approaches with D4-6 to experiment because I know D4's answer rules supreme and there would be no reason to use any other MVs)

PART E::

---------------------------------------------------------------------------------------------------------------------

1) Compression.

Compress part A - 1:

For A-1 I decided to compress the index here, because these attributes of the keys being used compress better since their are less of them and they are just key numbers.

2.2::

est: 18449.306641

run; 30.869142

estimation is the exact same but real runtime is quicker by about 15 seconds from the other index.

So I got the same estimated run time as the non-compressed version and it only uses 123.00. where uncompressed it uses 148.00

Compress part B - 1:

both the compressed version and the non-compressed version of my 4.2 general index are 0.97mb. There is no improvement in size here. The run time and estimated times of the compressed version were:

est: 148.431564

run: 1.673702

and the non-compressed being:

Q4.2:

est:171.275604

run: 1.330858

It was an improvement in estimation but not an improvement in actual runtime. This may have been caused by having to calculate the aggregate and needing to decompress quicky in place to do the math.

Compress part C - 1:

I'm going to add compression to the first flights materialized view and see how it works out:

"""

CREATE TABLE MV2\_ALL\_COMP\_E3 AS (SELECT sum(lo\_revenue) as total\_sum, d\_year, p\_brand1, s\_region, p\_category from lineorder, part, supplier, dwdate where lo\_orderdate = d\_datekey and lo\_suppkey = s\_suppkey and lo\_partkey = p\_partkey group by d\_year, p\_brand1, s\_region, p\_category) DATA INITIALLY DEFERRED REFRESH DEFERRED

CREATE INDEX MV2\_ALL\_INDEX\_COMP\_E3 ON MV2\_ALL\_COMP\_E3 (d\_year, p\_brand1) CLUTSER

"""

and I changed the queries to run under this as:

"""

select total\_sum, d\_year, p\_brand1 from MV2\_ALL\_COMP\_E3 where p\_category = 'MFGR#12' and s\_region = 'AMERICA' order by d\_year, p\_brand1

select total\_sum, d\_year, p\_brand1 from MV2\_ALL\_COMP\_E3 where p\_brand1 between 'MFGR#2221' and 'MFGR#2228' and s\_region = 'ASIA' order by d\_year, p\_brand1

select total\_sum, d\_year, p\_brand1 from MV2\_ALL\_COMP\_E3 where p\_brand1= 'MFGR#2239' and s\_region = 'EUROPE' order by d\_year, p\_brand1

"""

This gave performance of:

Q2.1)

est: 267.501892

run: 1.583010

Q2.2)

est: 267.834259

run: 1.483025

Q2.3)

est: 267.493042

run: 1.489249

This gives the exact same estimated times back but a couple slightly slightly different runtimes. I blame the runtimes on db2 being slower perhaps at this point in time.

Compress part D - 2:

With D2 now optimized we can get in there and have enough space now to fit the last flight. The design before was 4 indexes for each foreign key and then 3 indexes specifically for flight 1, 3, and 4. Now with compression we have the space to fit an index for flight 2 in there! (see D2 specifically it even has the list of flight 2 index I would use.) We now have covering indexes for every flight that are compressed. This will make for excellent design if the query optimizer chooses correctly!

Compress part D - 5:

\*\*\*If I were to compress all the MVs the space savings would be neglegable.\*\*\* (from before)

Now that I can compress all the materilized views we are able to add indexes to them. These indexes will allow faster joining and retrieval for all the queries.

"""

CREATE TABLE MV1\_COMPD5 AS (SELECT lo\_discount, lo\_quantity, lo\_extendedprice, d\_year, d\_yearmonthnum, d\_weeknuminyear from lineorder, dwdate where lo\_orderdate = d\_datekey) DATA INITIALLY DEFERRED REFRESH DEFERRED COMPRESS YES

refresh table DB2INST1.MV1\_COMPD5

CREATE TABLE MV2\_COMPD5 AS (select lo\_revenue, d\_year, p\_brand1, p\_category, s\_region from lineorder, dwdate, part, supplier where lo\_partkey = p\_partkey and lo\_suppkey = s\_suppkey and lo\_orderdate = d\_datekey) DATA INITIALLY DEFERRED REFRESH DEFERRED COMPRESS YES

refresh table DB2INST1.MV2\_COMPD5

CREATE TABLE MV3\_COMPD5 AS (select lo\_orderdate, lo\_revenue, lo\_custkey, s\_nation, s\_region, s\_city from lineorder, supplier where lo\_suppkey = s\_suppkey) DATA INITIALLY DEFERRED REFRESH DEFERRED COMPRESS YES

refresh table DB2INST1.MV3\_COMPD5

CREATE TABLE MV4\_COMPD5 AS (select s\_nation, lo\_revenue, lo\_supplycost, lo\_orderdate, lo\_partkey, c\_nation, c\_region, s\_region, c\_city, s\_city from lineorder, customer, supplier where lo\_custkey = c\_custkey and lo\_suppkey = s\_suppkey) DATA INITIALLY DEFERRED REFRESH DEFERRED COMPRESS YES

refresh table DB2INST1.MV4\_COMPD5

"""

flight 1: 200mb

flight 2: 300mb

flight 3: 300mb

flight 4: 600mb

Altogether the query doesn't show that we are saving any space at all(upon the query it shows both MV versions next to each other with the exact same mb used as the uncompressed versions). I suspect that db2 might be reporting incorrectly. So lets say it does save us some space that db2 is hiding. I would then create 1 or 2 indexes depending on the savings on the slowest flight so that it could be used for its queries to speed everything up.