COL362 Assignment 2

Aryan Dua - 2020CS50475, Divyanshu Agarwal - 2020CS10343

March 2023

1 Query 1

1.1 Picasso Query Template

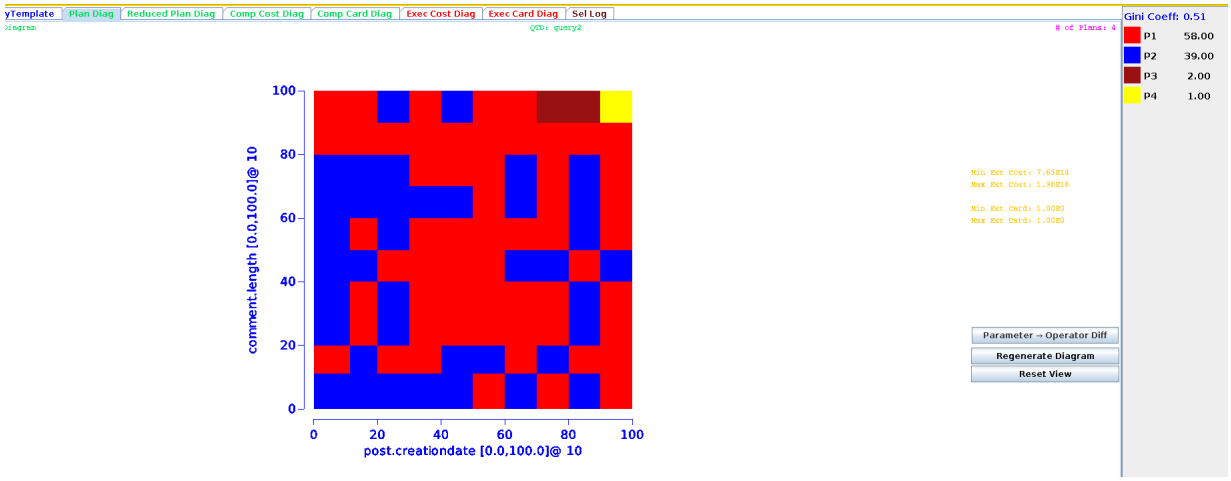
Here is the query template that we used for generating the diagrams in Picasso:

```
with
t1 AS
(
    SELECT id AS tid
    FROM tag
    WHERE tag.name in ('Frank_Sinatra' , 'William_Shakespeare' , 'Elizabeth_II' ,
    'Adolf_Hitler' , 'George_W._Bush')
),
t2 AS
(
    SELECT personid , person_hasinterest_tag.tagid as tag
    FROM person_hasinterest_tag
    WHERE person_hasinterest_tag.tagid in (select tid from t1)
),
filtered_posts AS
(
    SELECT id
    FROM post
    WHERE post.creationdate :varies
),
filtered_comments AS
(
    SELECT id
    FROM comment
    WHERE comment.length :varies
),
filtered_msgs as
(
    select * from filtered_posts
    union
```

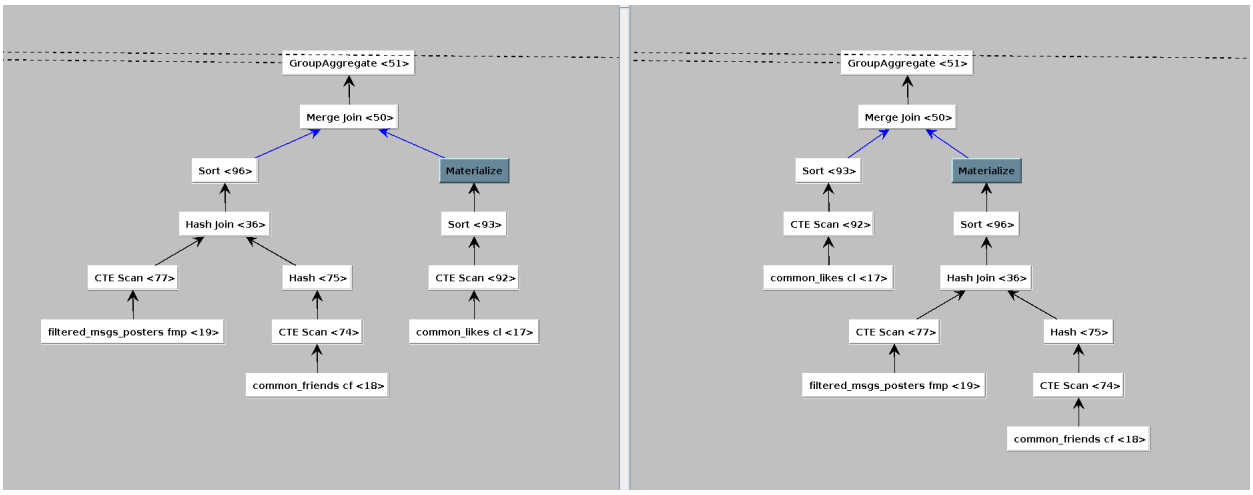
```
    select * from filtered_comments
),
messages AS
(
    SELECT id ,creatorpersonid AS pid FROM comment
    union
    SELECT id ,creatorpersonid AS pid FROM post
),
filtered_msgs_posters AS
(
    SELECT m.id AS id ,m.pid AS pid
    FROM filtered_msgs AS fm , messages AS m
    WHERE m.id=fm.id
),
person_likes_message AS (
    SELECT postid AS messageid ,personid FROM person_likes_post
    UNION
    SELECT commentid AS messageid,personid FROM person_likes_comment
),
msgs_with_likers AS (
    SELECT id as messageid,personid
    FROM person_likes_message AS plm ,filtered_msgs AS fm
    WHERE fm.id=plm.messageid
) ,
people_pairs AS
(
    SELECT t21.personid AS p1id, t22.personid AS p2id
    FROM t2 AS t21 ,t2 AS t22
    WHERE t21.personid<t22.personid and t21.tag = t22.tag
    group by p1id, p2id
    having count(t21.tag)>=2
),
common_friends AS
(
    SELECT f1.person1id AS fID , pp.p1id AS p1id, pp.p2id AS p2id
    FROM person_knows_person AS f1, person_knows_person AS f2, people_pairs AS pp
    WHERE (f1.person1id=f2.person1id AND f1.person2id=pp.p1id AND f2.person2id=pp.p2id) or
    (f1.person1id=f2.person2id AND f1.person2id = pp.p1id AND f2.person1id = pp.p2id )

    union all

    SELECT f1.person2id AS fID , pp.p1id AS p1id, pp.p2id AS p2id
    FROM person_knows_person AS f1, person_knows_person AS f2, people_pairs AS pp
    WHERE (f1.person2id=f2.person1id AND f1.person1id=pp.p1id AND f2.person2id=pp.p2id) or
    (f1.person2id=f2.person2id AND f1.person1id = pp.p1id AND f2.person1id = pp.p2id )
),
```

```
common_likes AS
(
    SELECT mwl1.messageid AS message_id , mwl1.personid AS p1id, mwl2.personid AS p2id
    FROM msgs_with_likers AS mwl1, msgs_with_likers AS mwl2, people_pairs AS pp
    WHERE mwl1.messageid=mwl2.messageid AND mwl1.personid=pp.p1id AND mwl2.personid = pp.p2id
),
common_likes_to_messages_by_common_friends AS (
    SELECT cl.p1id AS person1sid, cl.p2id AS person2sid ,
    count(cl.message_id) as mutualFriendCount
    FROM common_likes AS cl, common_friends AS cf, filtered_msgs_posters AS fmp
    WHERE cl.p1id = cf.p1id AND cl.p2id = cf.p2id AND cf.fID=fmp.pid AND cl.message_id=fmp.id
    GROUP BY cl.p1id , cl.p2id
    HAVING count(cl.message_id) >= 10
)
    select upper.person1sid as person1sid , upper.person2sid as person2sid ,mutualFriendCount
    from common_likes_to_messages_by_common_friends as upper
    where (person1sid,person2sid)not in (select pkp.person1id , pkp.person2id from
    person_knows_person as pkp)
    and (person2sid,person1sid)not in (select pkp.person1id , pkp.person2id from
    person_knows_person as pkp)
    ORDER BY person1sid asc, mutualFriendCount desc, person2sid asc ;
```

1.2 Plan Diagram



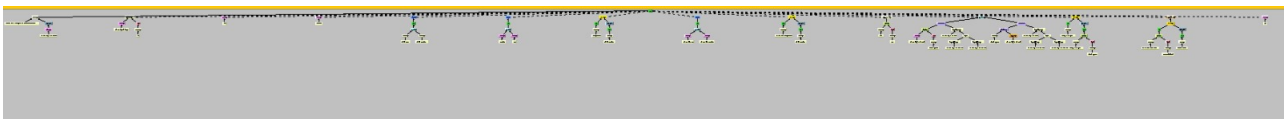
1.3 Comparison between top 2 plans



1.4 Observations

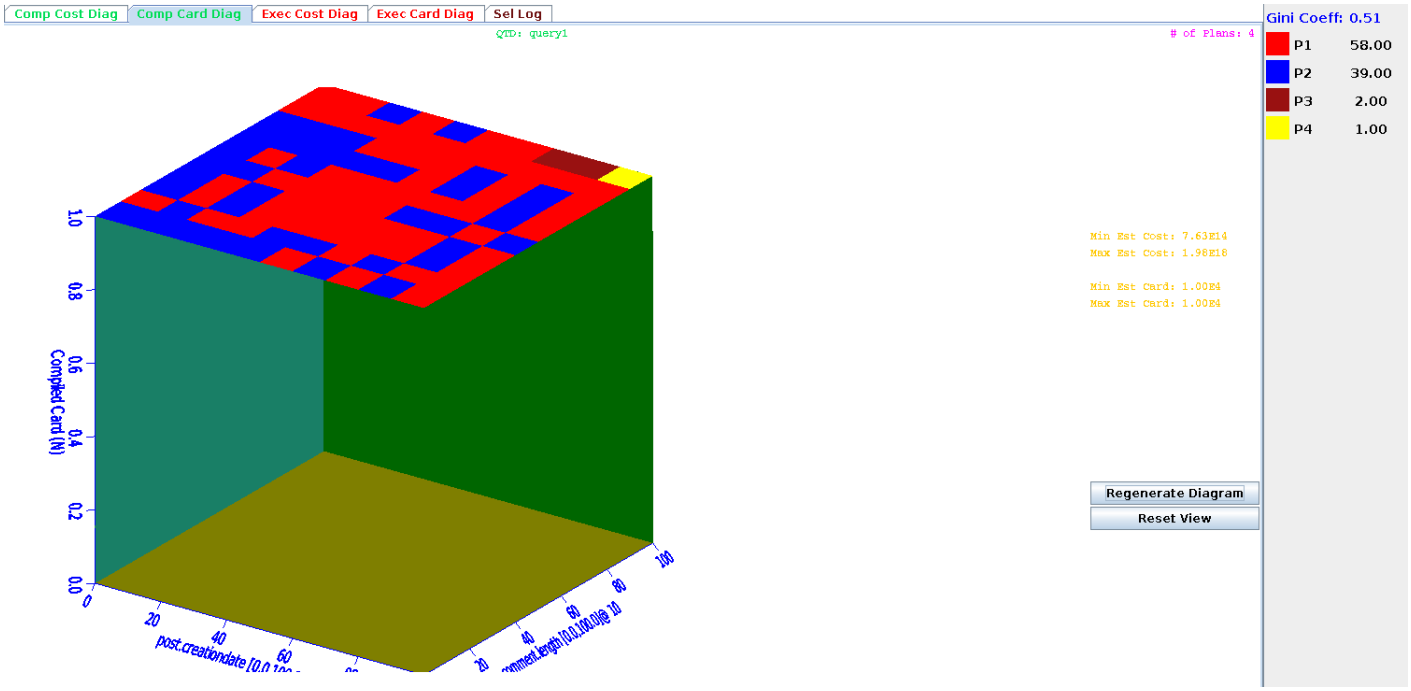
We can see that when either one of CreationDate or comment length has low selectivity then plan 2 dominates. Otherwise both plan 1 and plan 2 are equally present . Plan1 and 2 differ only on how they join the tables filtered message poster, common friends and common likes. While plan1 materialises common likes before merge join , plan2 materialises the sorted hash join of filtered messages first. When a table is materialised it takes space on the disk and hence the table with smaller number of records is decided to be materialised. when either of the two selectivity is low then the no. of records in hash join of filtered messates and common friends is lower than common likes and this is decided to be materialised.
Further we can see that when both of the selectivites are higher the plans P3 and P4 dominate.

1.5 Best Plan Tree

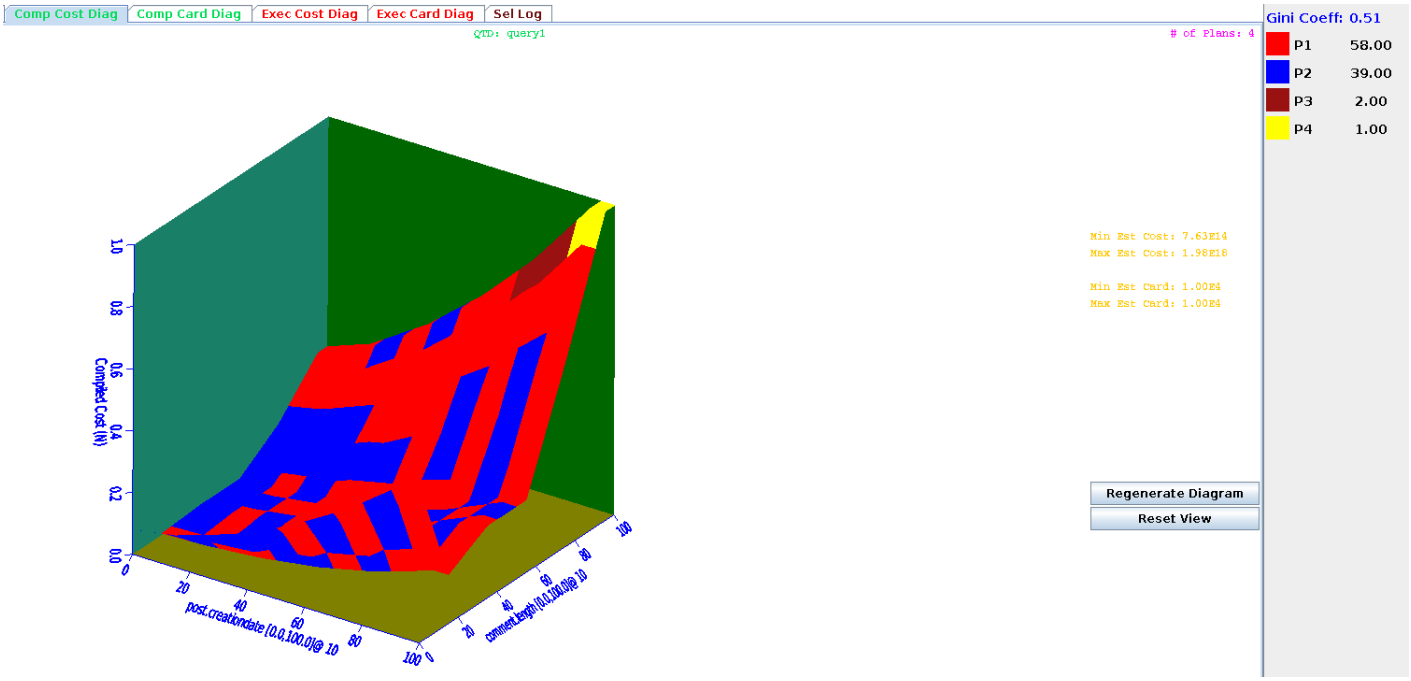


This is the P1 Plan tree for the query.

1.6 Cardinality Diagram



1.7 Compiled Cost Diagram



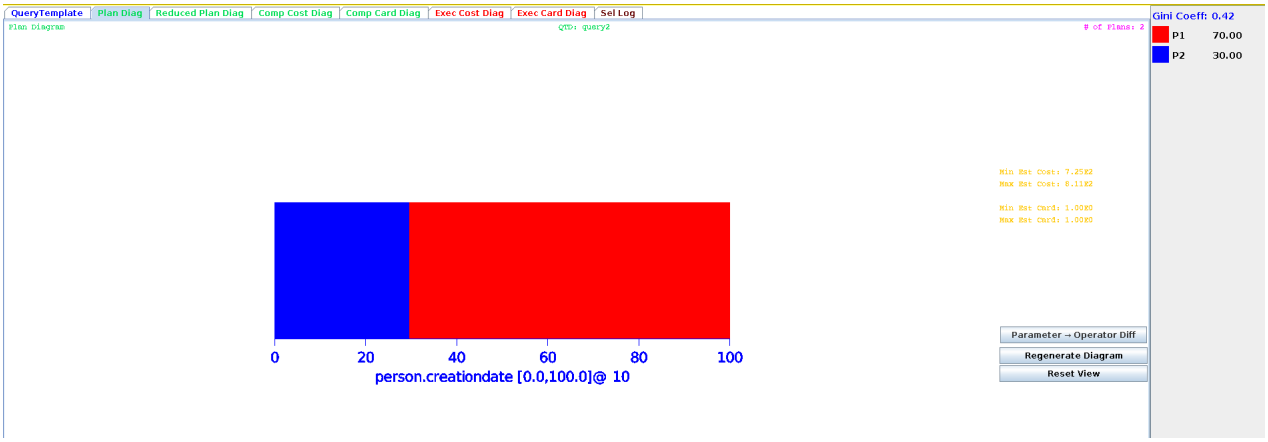
We can see that as the selectvity increase the compiled cost increases with both the parameters except a small fall when creationdate has 100 selectivity but comment length has low selectivity.

2 Query 2

2.1 Picasso Query Template

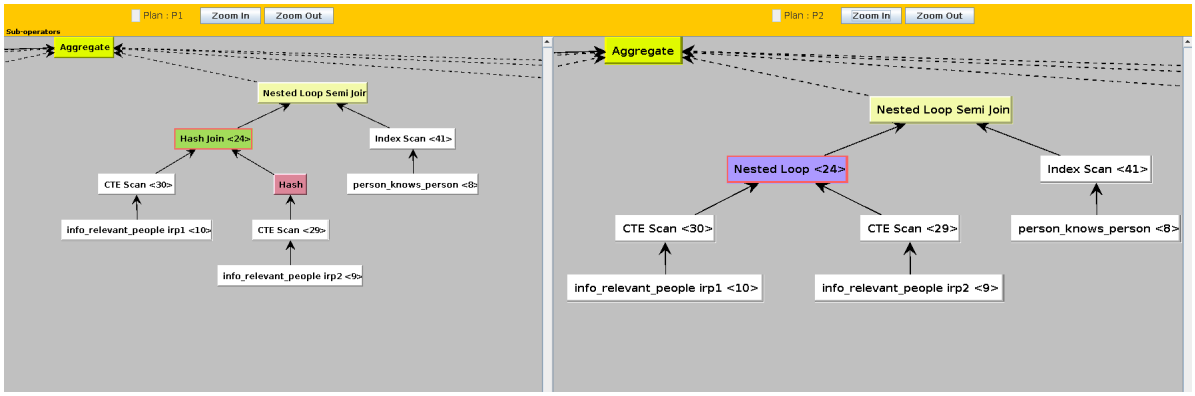
```
with
city_country as
(select place1.id as cityid, place2.id as countryid
from place as place1 , place as place2
where place1.type='City' AND place2.type='Country' AND place1.partofplaceid=place2.id),

id_of_country as
    (select id
    from place
    where place.type='Country' and place.name ='China'),

relevant_people as
(select p.id as pid
from person as p, city_country as cc ,id_of_country as idc
where p.locationcityid=cc.cityid and cc.countryid=idc.id and creationdate :varies)

,
info_relevant_people as (select rp.pid as pid ,SUBSTRING((CAST(birthday as text)),6,2) as birthmonth
from relevant_people as rp , person as p  ,  person_studyat_university as psu
where rp.pid = p.id AND rp.pid= psu.personid
) -- this table here contains all the relevant people with the info we need about them .

,
filtered_person_knows_person as
(select irp1.pid as person1id , irp2.pid as person2id
from  info_relevant_people as irp1 , info_relevant_people as irp2
where irp1.birthmonth=irp2.birthmonth and irp1.univid=irp2.univid and (irp1.pid,irp2.pid) in (select
,
transitive_triple as
(select pp1.person1id as p1id , pp1.person2id as p2id , pp2.person2id as p3id
from filtered_person_knows_person as pp1 JOIN filtered_person_knows_person as pp2
on pp1.person2id = pp2.person1id)
,
cliques as
(   select tt.p1id as p1id, tt.p2id as p2id, tt.p3id as p3id
    from transitive_triple as tt
    where (p1id,p3id) in (select person1id, person2id from filtered_person_knows_person)
)
select count(*) from cliques;
```

2.2 Plan Diagram



It is asked to vary two columns simulatneuosly however since picasso allows to vary only on column per table at a time I have varied only one.

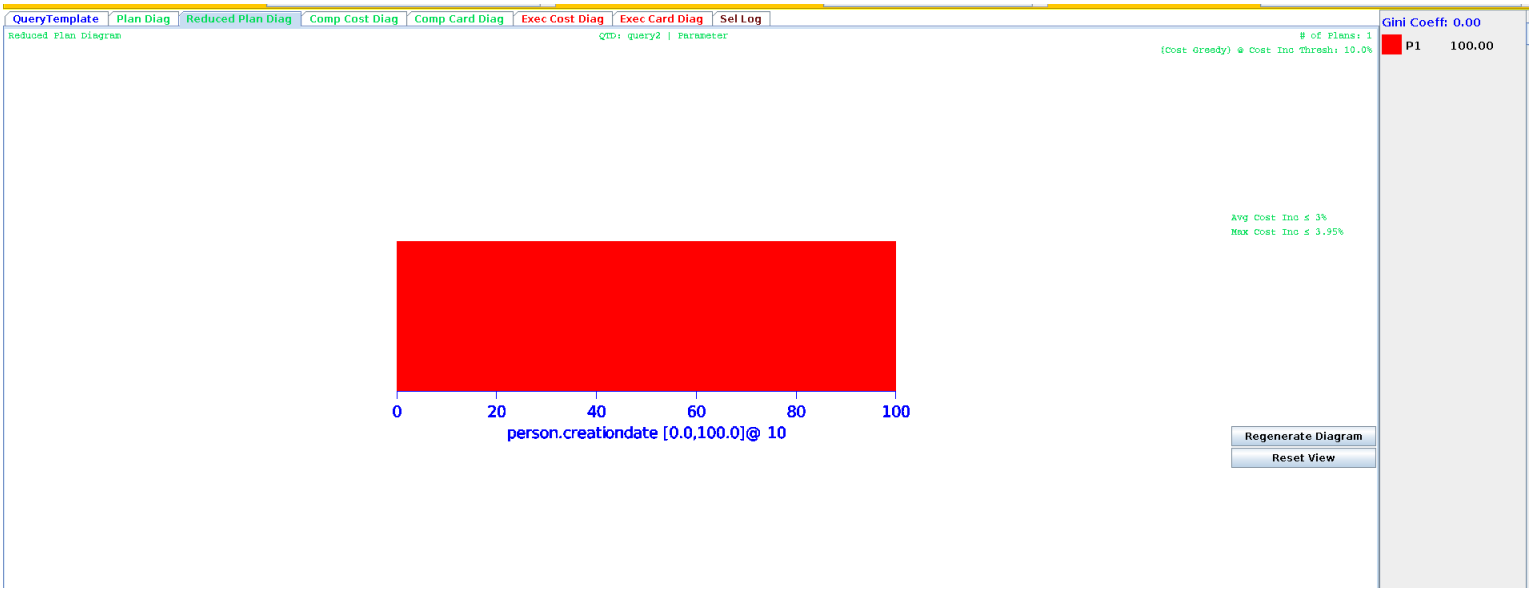
2.3 Comparison Between top 2 plans



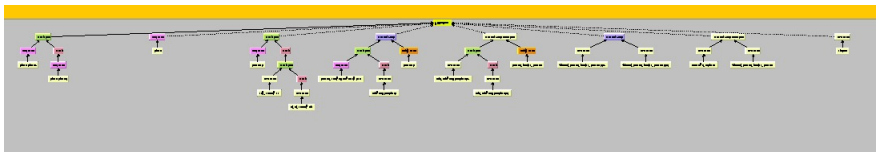
2.4 Observations

It can be seen that when the selectivity is high plan 1 is optimal else plan 2 is optimal. Plan 1 does a hash join of tables irp1 and irp2 while plan2 does a nested loop join of the tables. The irp tables are joined on birthmonth and univid , when the selectivity is high there is a lot of data and since we know hash join is typically faster than nested loop join when the size of the tables being joined is large and there are no suitable indexes available to improve the performance of nested loop join. So it is clear why Plan 1 performs better than plan 2 on high selectivity.

2.5 Reduced Plan Diagram

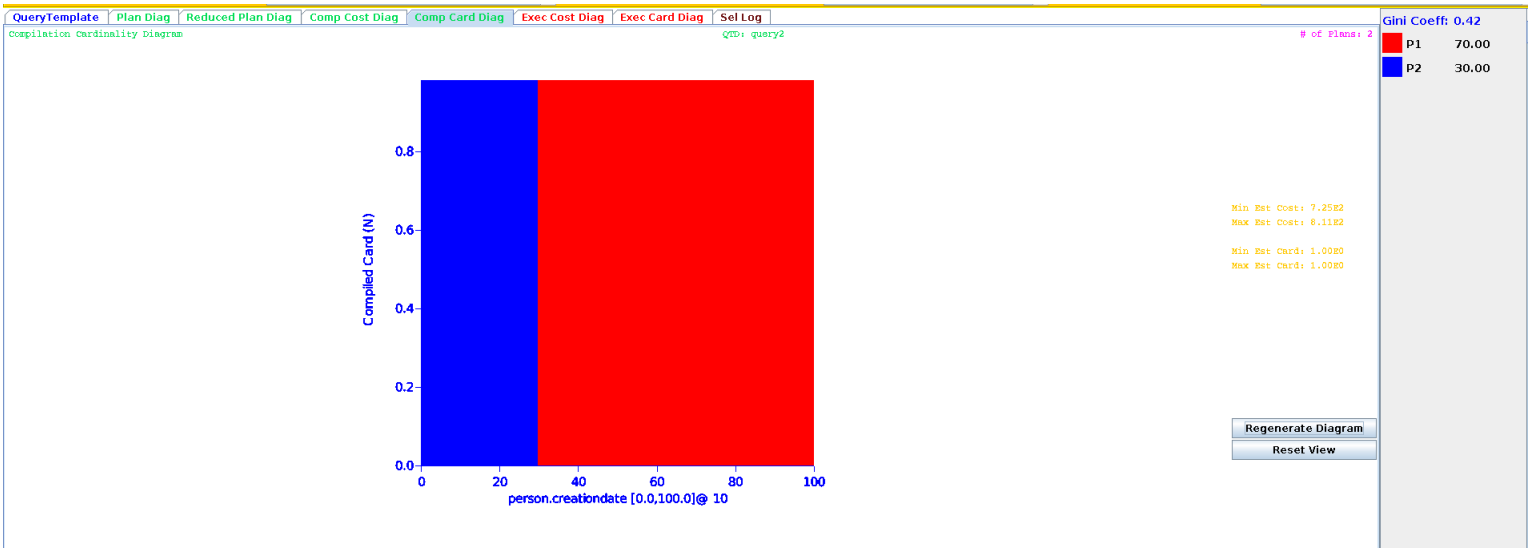


2.6 Best Plan Tree

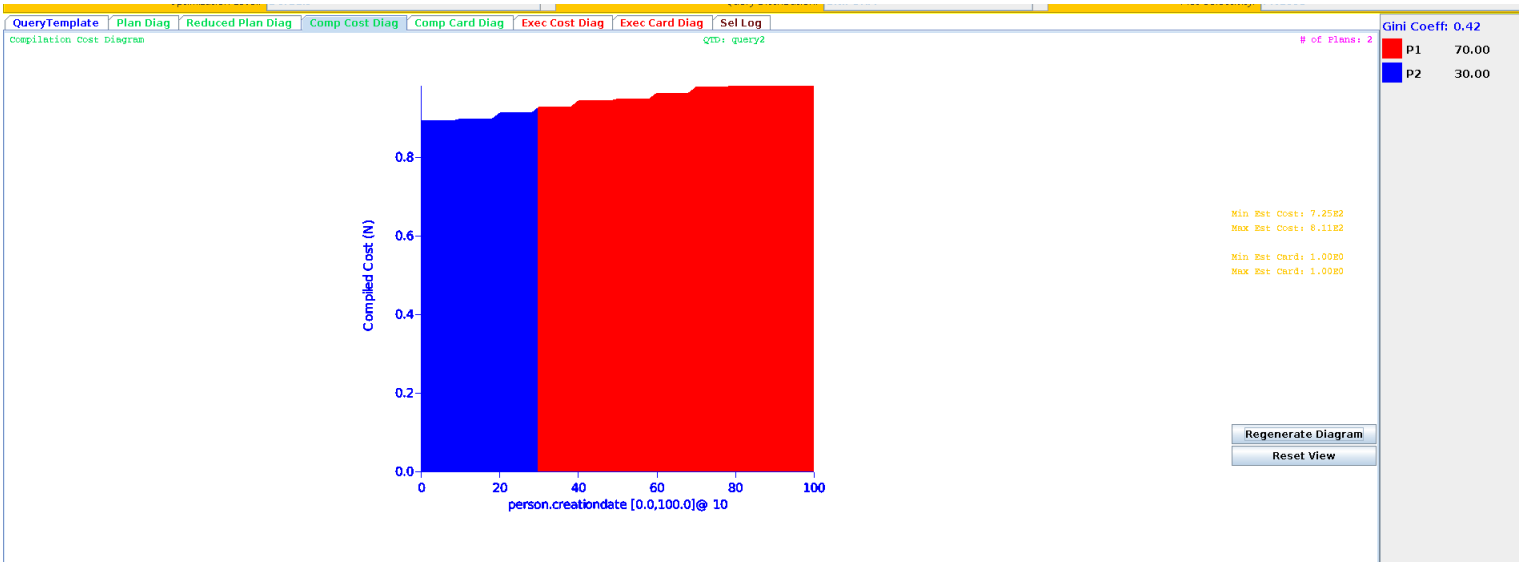


This is the P1 Plan tree for the query.

2.7 Cardinality Diagram

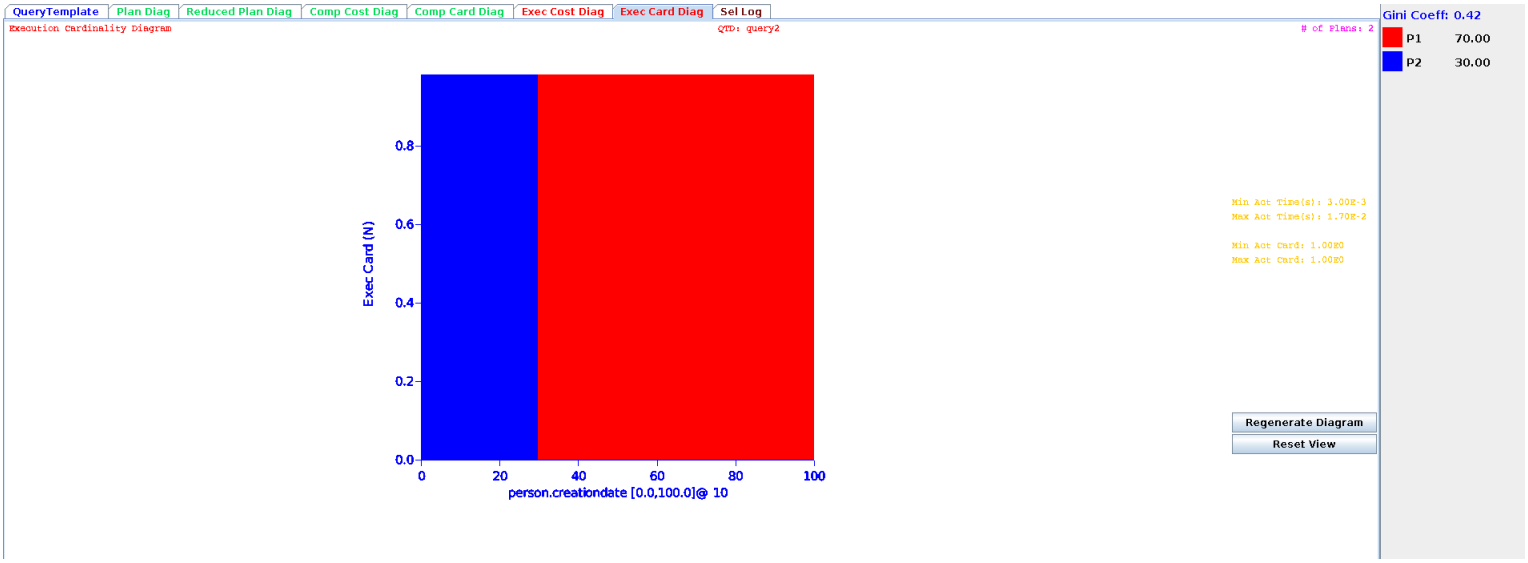


2.8 Compiled Cost Diagram

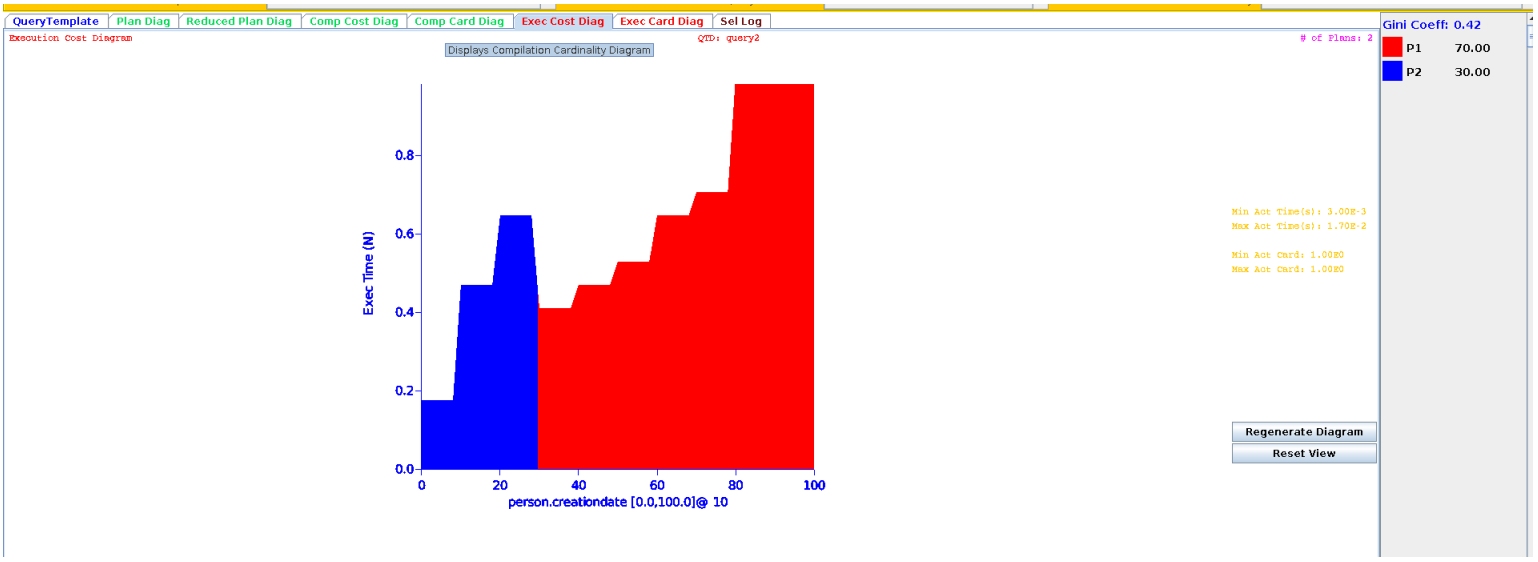


We can see that as the creationdate has more selectivty the compilation cost increase monotonically.

2.9 Execution Cardinality Diagram



2.10 Execution Cost Diagram



2.11 Observations

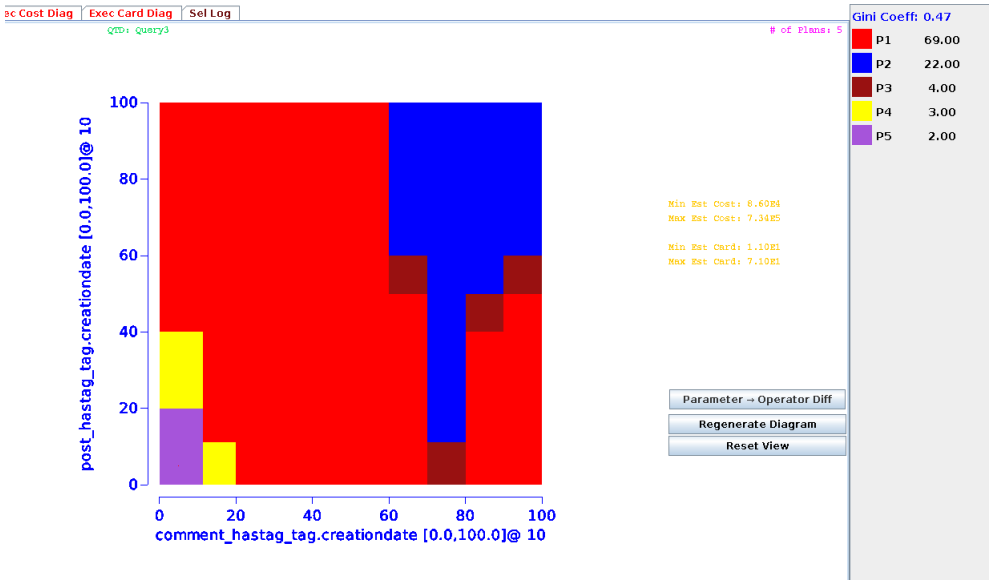
We can see that when we fix a plan then as selectivity increases the cost increases. This is expected as the more the selectivity the more the data is being worked with and the higher the cost. Now when we compare across plans we see that when we change our plan from plan 2 to plan1 near 30 % selectivty we have a reduction in cost, this is because in regions of high seelctivity data is larger so for selectivity 30 and more red plan does hash join as explained above and performs better with lower cost, while if the selectvity is lower the cost of hashing would exceed the speed up and hence nested join performs better .

3 Query 3

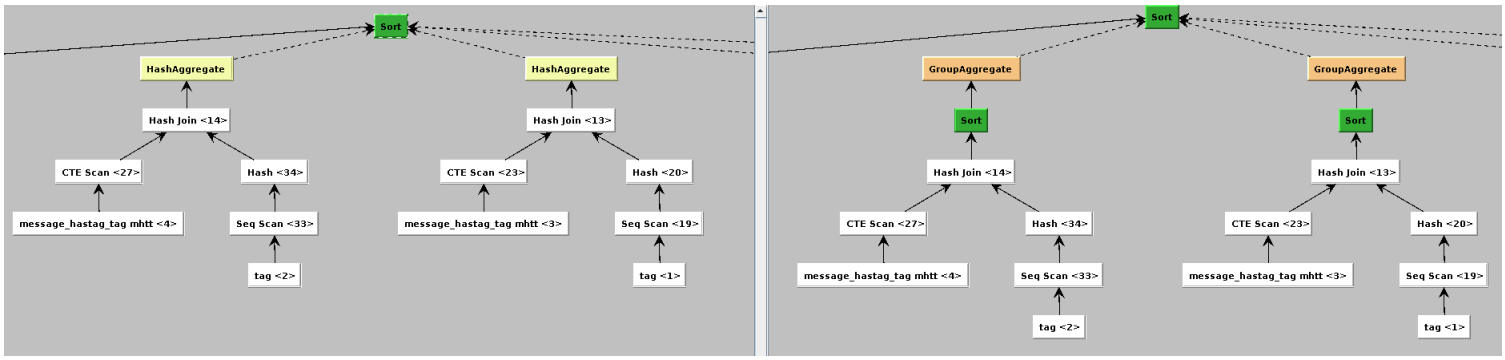
3.1 Picasso Query Template

```
with
message_hasTag_Tag as
(
    select creationDate as date , commentid as id, tagid as tagid
    from comment_hastag_tag
    where comment_hastag_tag.creationdate :varies

union

    select creationDate as date , postid as id, tagid as tagid
    from post_hastag_tag
    where post_hastag_tag.creationdate :varies
),
message_count_start as
(
    select tag.id as tagid, tag.name as name ,count(mhtt.id) as count
    from tag,message_hasTag_Tag as mhtt
    where mhtt.tagid=tag.id and mhtt.date <= '2010-12-03' and mhtt.date >='2010-02-03'
    group by tag.id, tag.name
),
message_count_end as
(
    select tag.id as tagid, tag.name as name ,count(mhtt.id) as count
    from tag,message_hasTag_Tag as mhtt
    where mhtt.tagid=tag.id and mhtt.date >= '2010-12-03'  and mhtt.date <='2011-05-03'
    group by tag.id, tag.name
),
tags_selected as
(
    select mce.name as tname , mce.tagid as tagid , mcs.count as mcs , mce.count as mce ,
    t.TypeTagClassId as classid
    from message_count_end as mce , message_count_start as mcs , tag as t
    where t.id=mce.tagid and  mce.tagid=mcs.tagid and mcs.count>=5*mce.count
)
select tc.name as tagclassname , count(tagid) as count
from tags_selected as ts, tagclass as tc
where tc.id = ts.classid
group by tc.name
order by count desc , tagclassname asc;
```

3.2 Plan Diagram



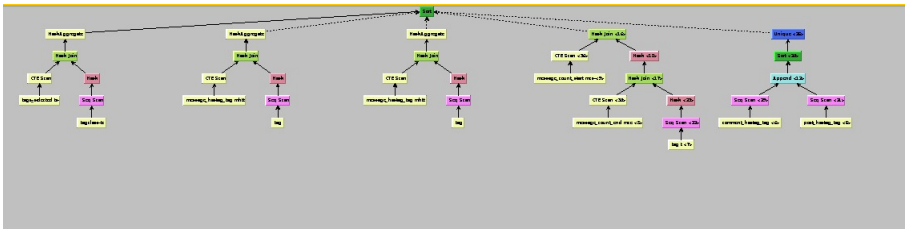
3.3 Comparison between top 2 plans



3.4 Observations

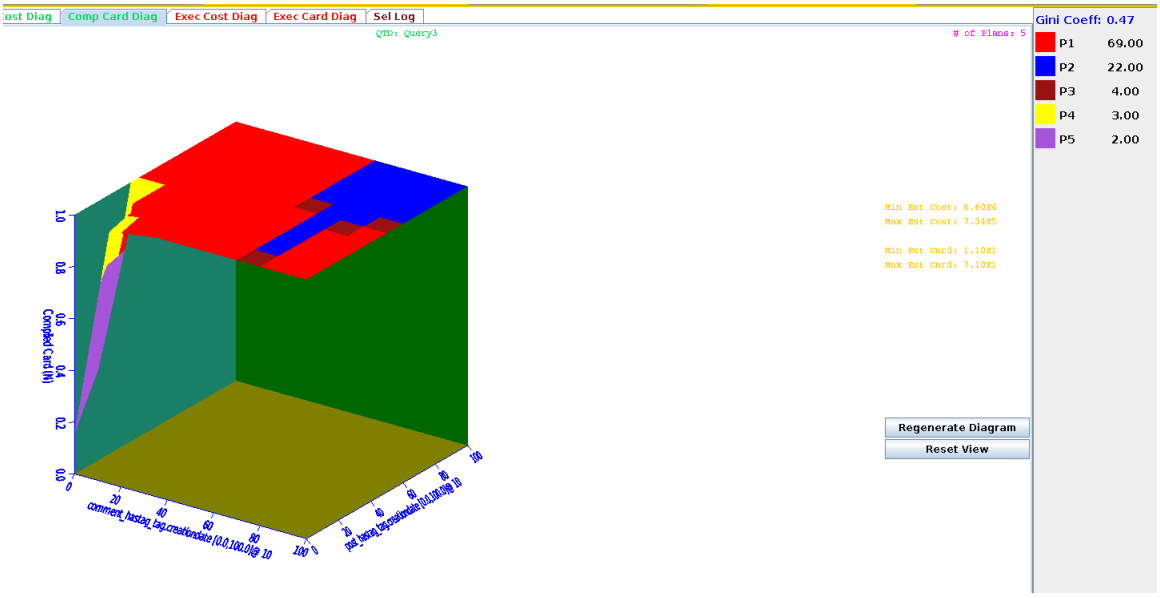
We can see that when we have low selectivity over creationdate of comments then plan 1 performs better . But then we can see a sudden change at 60-70 % selectivity after which plan 2 performs better. This is indicates the presence of the middate variable. As soon as the comment and post creation date is beyond middate we would have entries in the message count end table which would lead to presence of entries for the final tag selection (before this tag selection was trivial as message count end was always 0) . In order to make the tag selection more efficient the Plan 2 uses group aggregates while Plan1 does not need group aggregates since count is already zero for message count end. Due to this plan 2 performs better than plan1 since it uses group aggregates.

3.5 Best Plan Tree

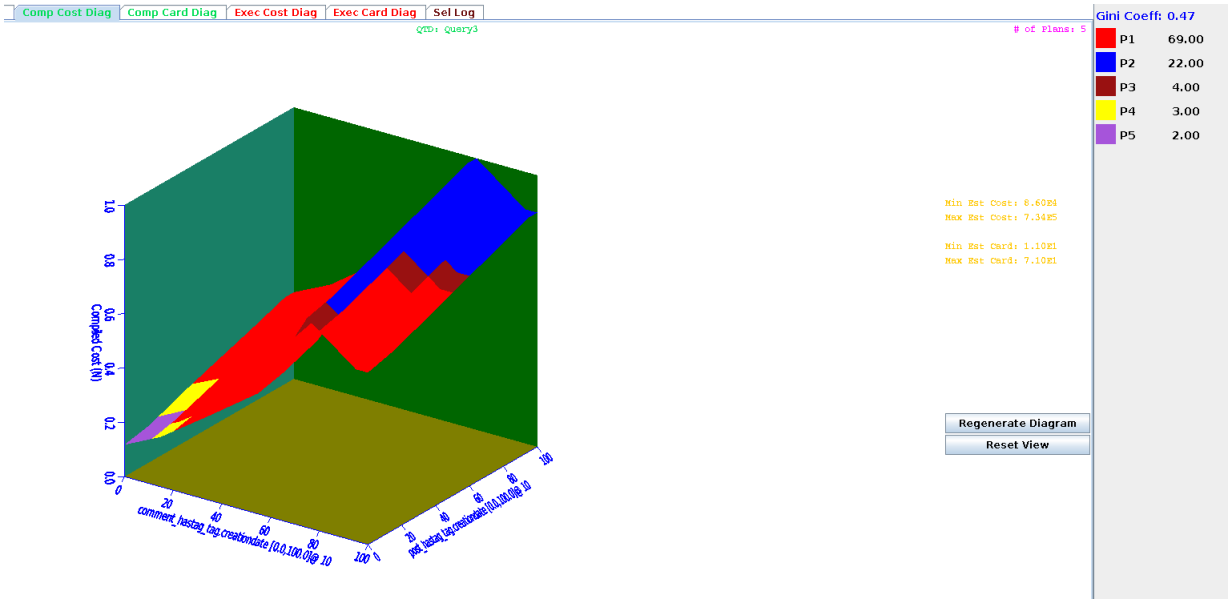


This is the P1 Plan tree for the query.

3.6 Cardinality Diagram



3.7 Compiled Cost Diagram



We can see that as long as the red plan is being used the cost increases with increasing creation date. Then once the creationdate crosses middate we see a sudden decrease in cost due to the change in plan as described in the observation above.

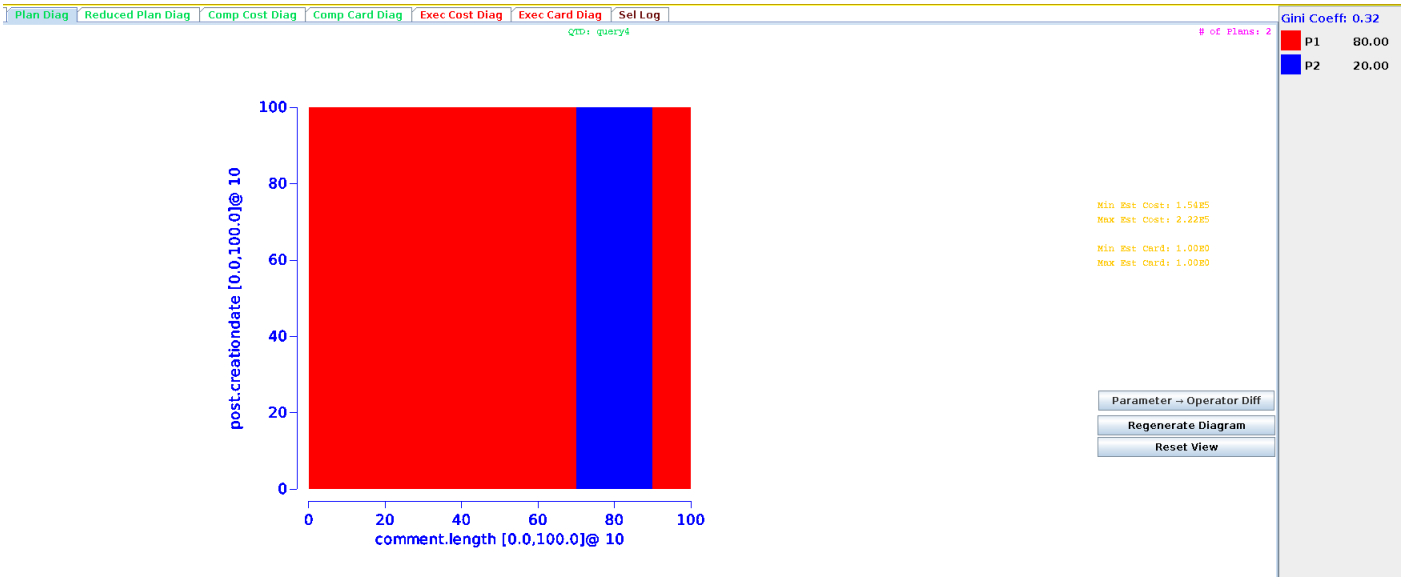
4 Query 4

4.1 Picasso Query Template

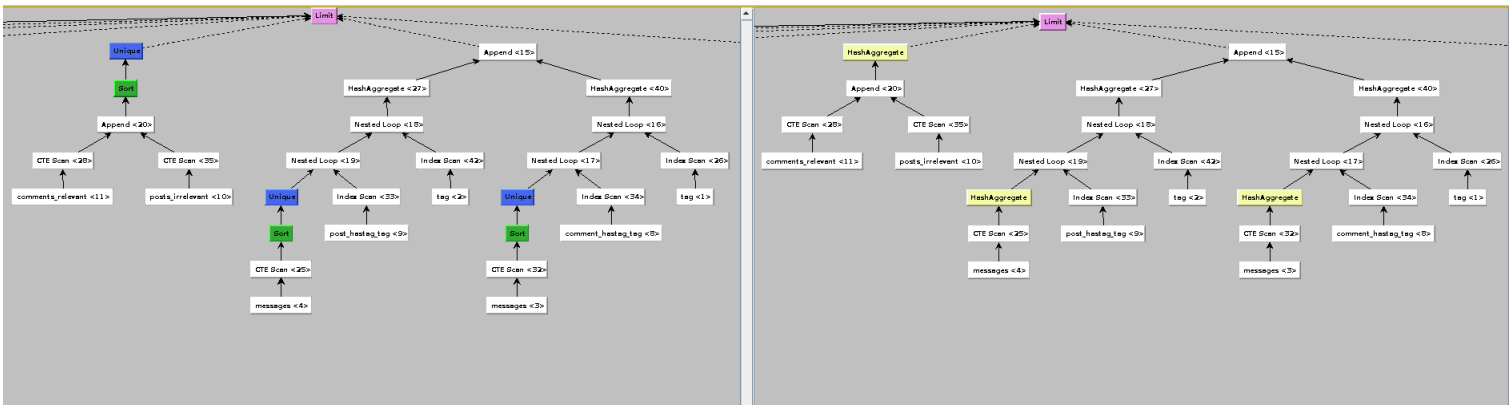
```
with
new_comment as
(
    select * from comment
    where comment.length :varies
),
comments_relevant as
(
    select parentcommentid as messageid
    from new_comment
    where parentpostid is null
    group by parentcommentid having count(id)>=4
),
posts_relevant as
(
    select parentpostid as message_id
    from new_comment
    where parentcommentid is null
    group by parentpostid having count(id)>=4
),
posts_irrelevant as
(
    select message_id
    from posts_relevant,post
    where post.id=message_id and post.creationdate :varies
),
messages as
(
    select * from comments_relevant
    UNION
    select * from posts_irrelevant
),
 message_hastag_tag as
(
    select count(postid) as c  , tagid , tag.name as tagname
    from post_hastag_tag , tag
    where tag.id=tagid and post_hastag_tag.postid =postid and postid in (select * from messages)
    group by post_hastag_tag.tagid,tag.name

    union all

    select count(commentid ) as c  ,tagid , tag.name as tagname from comment_hastag_tag ,tag
    where tag.id=tagid and comment_hastag_tag.commentid =commentid
```

```
    and commentid in (select * from messages)
    group by comment_hastag_tag.tagid,tag.name
)
 select tagname,sum(coalesce(c,0)) as count
 from message_hastag_tag
 group by tagname
 order by count desc , tagname asc
 limit 10;
```

4.2 Plan Diagram



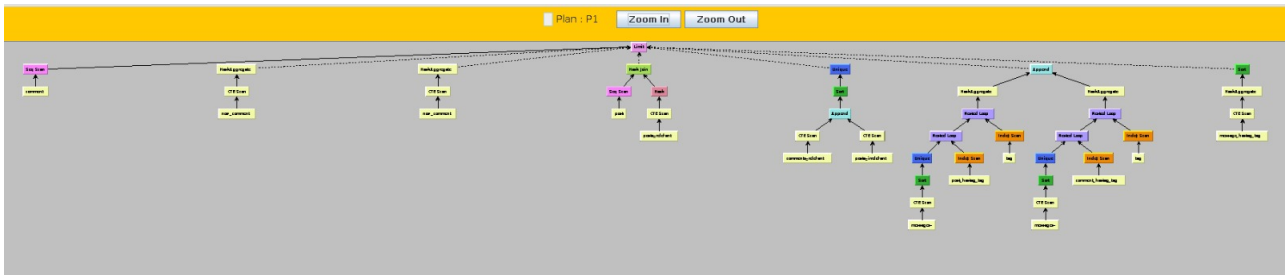
4.3 Comparison between top 2 plans



4.4 Observations

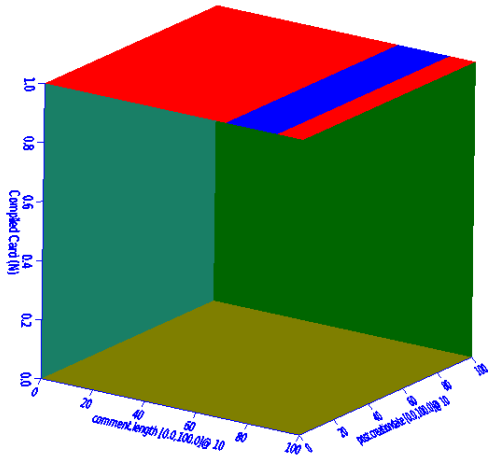
We can see that the optimal plan choice is independent of the the selectvity on post creationdate, this can be exxplained by the fact that the plans differ only on sorting of the tables before joins. Plan1 sorts and finds unique elements while plan2 does hash aggregate. Plan1 performs better in low selectivty due to the presence of lesser number of entries making sorting faster while plan 2 performs better on large selectvity since hashing performs better when data size is large.

4.5 Best Plan Tree

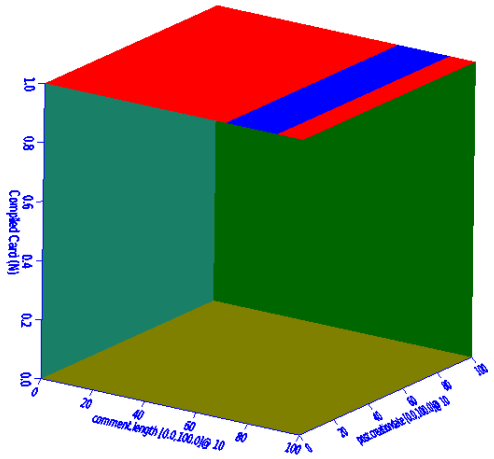


This is the P1 Plan tree for the query.

4.6 Cardinality Diagram



4.7 Compiled Cost Diagram



We can see that the compilation cost is independent of the selectivity of comment length and creation date.

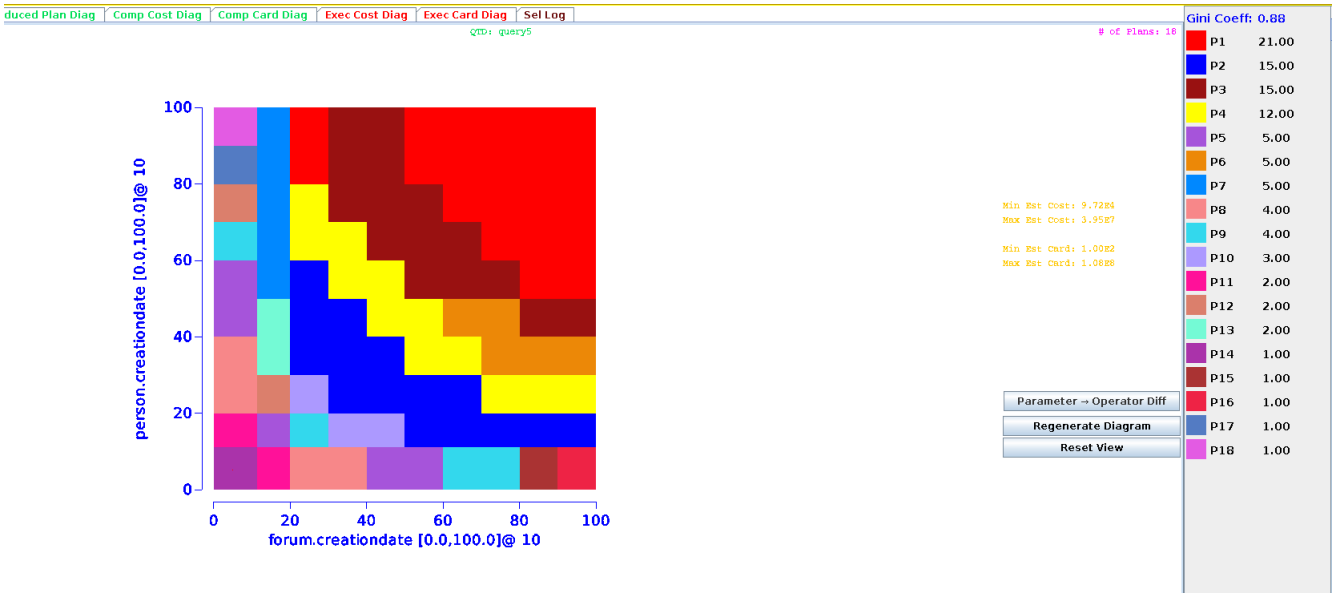
5 Query 5

5.1 Picasso Query Template

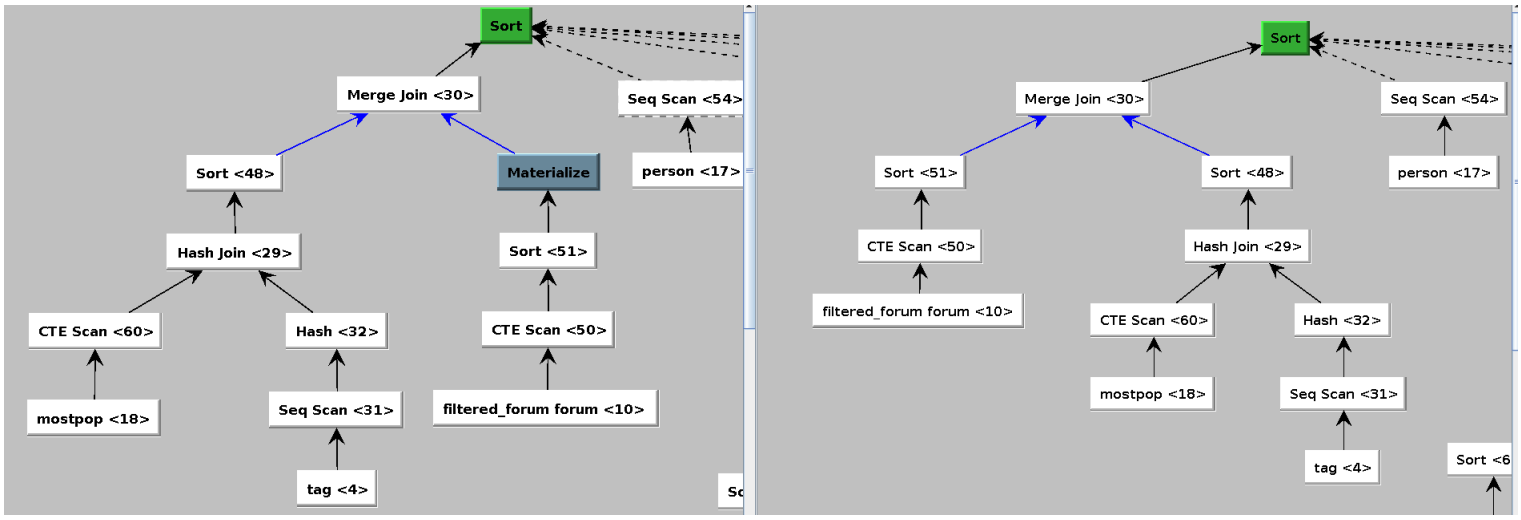
```
with
filtered_forum as
(
    select forum.id, forum.moderatorpersonid , forum.title
    from forum
    where forum.creationdate :varies
),
filtered_person as
(
    select id , locationcityid
    from person
    where person.creationdate :varies
),
test_table(num_posts, forum_id, tagid, rank) as
(
    select num_posts, forum_id, tagid, ROW_NUMBER() OVER (PARTITION BY forum_id
    ORDER BY num_posts desc) as rank
    from
    (
        select count(post.id) as num_posts, forum_ids as forum_id, tagid
        from
        (
            select condition1 as forum_ids
            from
            (
                select forum.id as condition1
                from place as t1, place as t2, filtered_person as person, filtered_forum as forum
                where t1.type = 'City' and t1.partofplaceid = t2.id and t2.name = 'India'
                and t1.id = person.locationcityid and forum.moderatorpersonid = person.id
            ) as t3,
            (
                select distinct containerforumid as condition2
                from post_hastag_tag, post, tag, tagclass
                where post.id = postid and tag.id = tagid and TypeTagClassId = tagclass.id
                and tagclass.name = 'TennisPlayer'
            ) as t4
            where condition1 = condition2
        ) as t5, post, post_hastag_tag
        where post.containerforumid = forum_ids and post_hastag_tag.postid = post.id
        group by forum_ids, tagid
    ) as t6
),
max_count(num_posts, forum_id) as
```

```
(
    select num_posts, forum_id from test_table
    where rank = 1
),
mostpop(num_posts, forum_id, tagid) as
(
    select test_table.num_posts, test_table.forum_id, tagid
    from test_table, max_count
    where test_table.num_posts = max_count.num_posts and test_table.forum_id = max_count.forum_id
)
select forum_id as forumid, forum.title as forumtitle, tag.name as mostpopulartag,
num_posts as count
from mostpop, filtered_forum as forum ,tag
where forum.id = forum_id and tagid=tag.id
order by count desc, forumid, forumtitle, mostpopulartag;
```

5.2 Plan Diagram



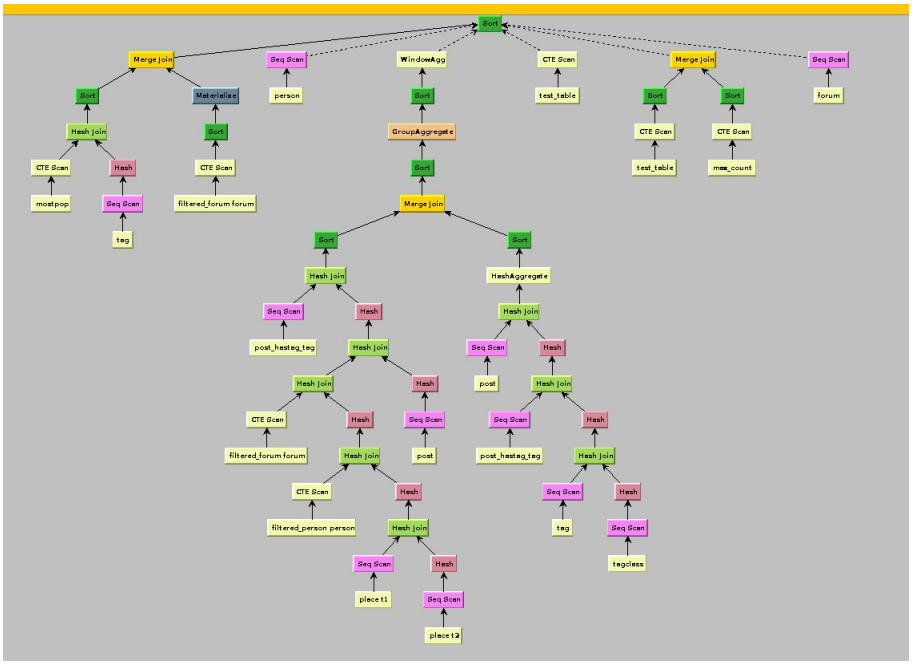
5.3 Comparison between top 2 plans



5.4 Observations

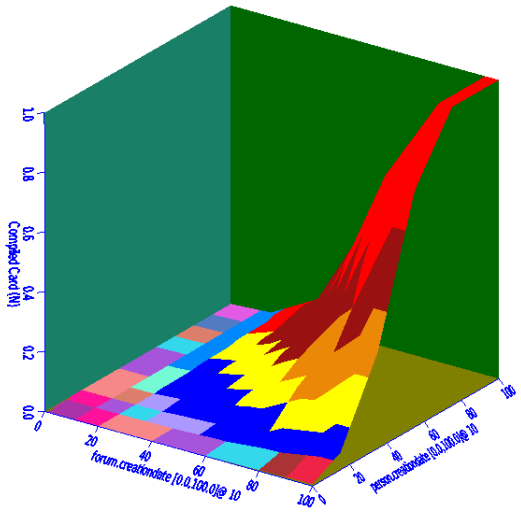
We can see a lot of plans. I have compared the top2. We can see in regions of high selectivty plan 1 performs better while in region of low selectivty plan 2 performs better. This is because plan1 materialises the sorted forum table so the operations do not needed to be perform again for the merge join. When the data is small merge join can do sorting on the go, but when it is large the materialised data make the merge join fast.

5.5 Best Plan Tree

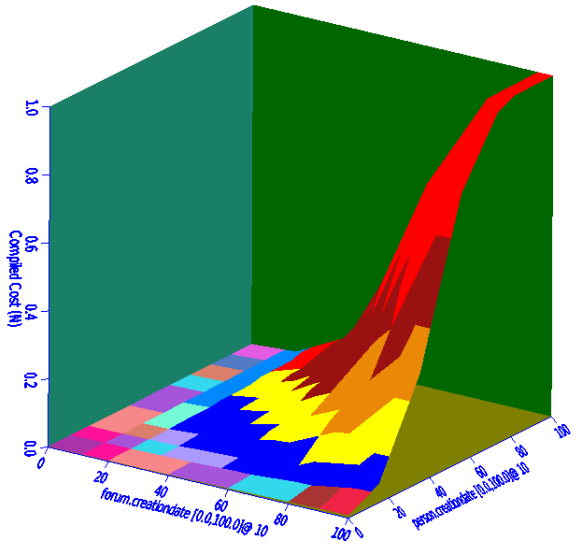


This is the P1 Plan tree for the query.

5.6 Cardinality Diagram



5.7 Compiled Cost Diagram



We can see the compiled cost increase monotonically as the selectivity increases this is due to more data being present. We can see change in gradient when the plan changes.