Aleksas Murauskas

COMP 421 Database Systems

User:

10,000 users in system

Average Size=

Attraction:

7500 Attractions

Avg Size:

Type: 10 different types of attraction

Review:

On average, 3 per user

Avg Size:

Rating: Can have 1,2,3,4,5. Uniform distribution-> each has a 20% chance.

Date is uniformly distributed over 365 days in the year. Each date has a 1/365 chance

Ex. 1

A. Find the number of data pages to store each of the three tables, User Attraction and Review

User: Pages needed =

Attraction: Pages needed =

Review: Pages needed =

B. What is the I/O cost of the query:

Select \* From Review WHERE rating =X AND date >Y

Basic Information:

Theoretical amount of matching tuples:

On rating:

On date:

1. In the case of: No index

Since there is no index, a matching relation can be found on any data page, therefore all data pages must be read. Thus,

1. In the case of: unclustered type I index on rating

Leaf pages when indexed on rating

Distinct values of Ratings =5,

Bytes per data entry =

Data entries per page = data entries per page

Leaf pages =

Estimated 20% of tuples will match X

20% of leaves will be used:

In worst case since pages will be recalled, a data page will be called for each matching tuple ,

Thus,

1. In the case of: unclustered type II index on date

Leaf pages when indexed on date

Distinct values of date = 365,

Bytes per data entry =

Data entries per page =

Leaf pages =

COST=

Ex.2 unclustered type II index on attractionid on Attraction

There will be 30,000 tuples since every review is related to exactly one amount

Information:

Building an unclustered type 2 index on attractionid:

Data pages to store attraction= 225 (found in question 1)

Distinct values of attractioid=tuples of attraction, therefore on record per value of attractionid

Bytes per data entry =

Data entries per page:

leaves

QA. Index nested loop join between Attraction and Review

For each tuple r in Review do

Find all matching tuples a in Attraction through index

Then add all <u,g> to result

On average there are

Cost of an Index nested loop = Review pages +CARD(review) \*(cost of finding matching tuples)

Attractions per review = 7500/30000 = .25

OR

Returned Tuple estimate: 30,000 tuples assuming every attraction ID within Review Is valid

Q2. block nested loop join between Attraction and Review and Attraction is the outer relation

Given 50 free buffer pages:

Therefore

COST:

Cost=

Q3. Sort merge between Attraction and Review (not already sorted)

attraction sort costs:

225 pages/ 50 buffer = 5 runs, cost = 225

Review sort costs:

2890 pages/ 50 = 58 runs, cost: 2\* 2890

Passes needed: 1+

Pass 2 Needed for sure on review

Pass 2:

Cost = runs pages\* 2098,

+ pipeline the output of each pass to the join operator, then perform the join.

Reduces the costs.

COST:

Ex. 3

SELECT U.username, R.rating, R.comment

FROM User U, Review R, Attraction A

WHERE U.username = R.reviewer

AND R.attractionid = A.attractionid

AND R.rating >= 4

AND A.type = 'hotel'

QA. Optimize the expression according to the rules discussed in class

First we will apply the query requirements to the individual tables

Then we do the following joins.

Allow Capital X to represent join

Case 1

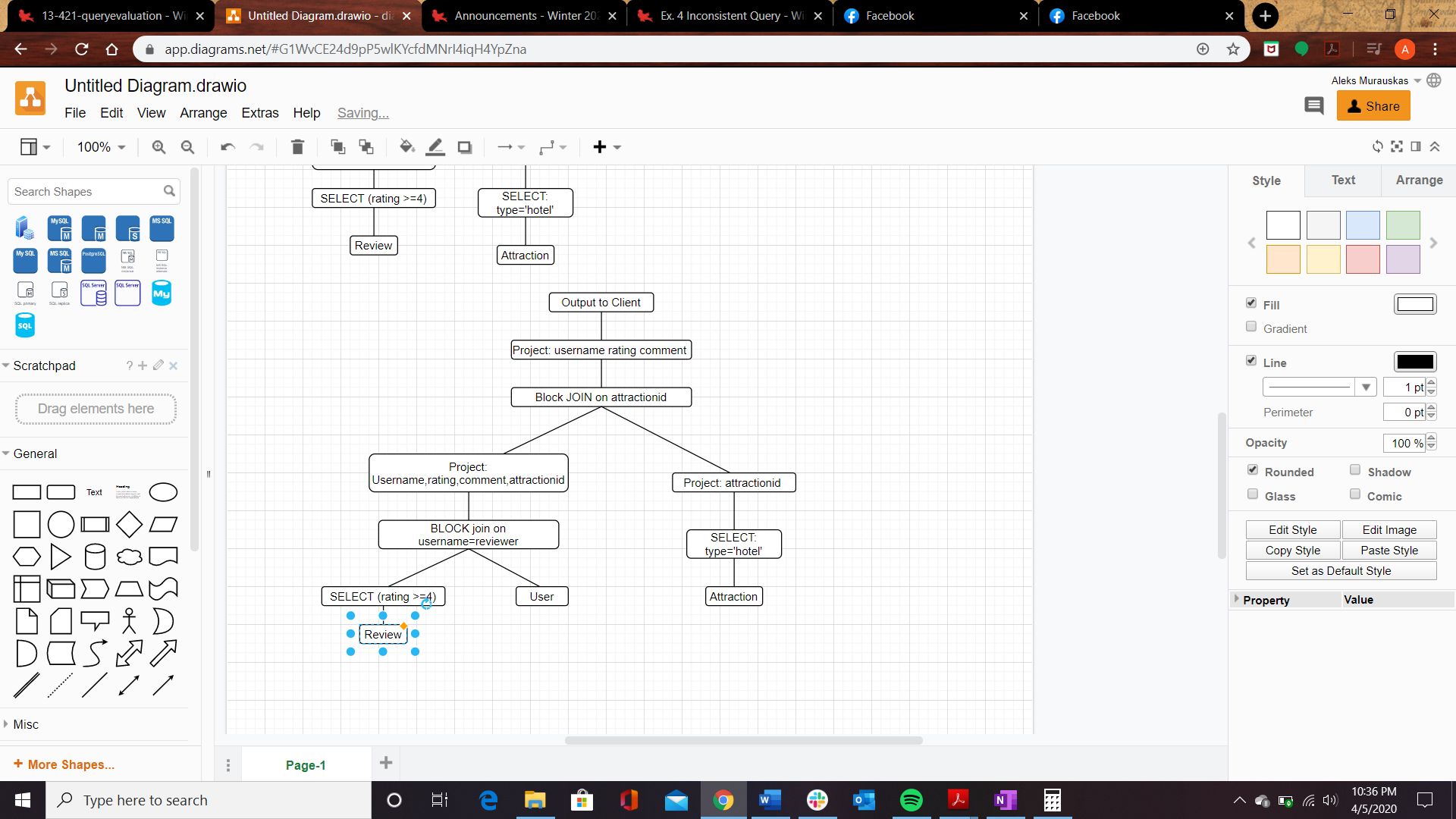
Another Way is

Case 2:

QB.

Reduction of rating >=4 is 4 an 5 therefore 40%

In case 1:



We join User and Review first

Determine which is better to use Index or Block joins

Block:

Now with the index on review

Number of reviews per User =3

Number of leaves holding

Of these options Block with User on the inside is the optimal, IO cost = 10,288

Our output is temp T, It holds 12,000 records

It is then projected with username, rating, comment and attractionid. New length = 12+ 8+ ¾(300)+8=253 bytes. This will occupy , this much is too large so it must be written to disk. Incurr IO 759 cost.

Next we join with attraction:

We run select on attraction: reduceing the tuples by 1/10 leaving

If we project just the attraction ids, we can store less in main memory.

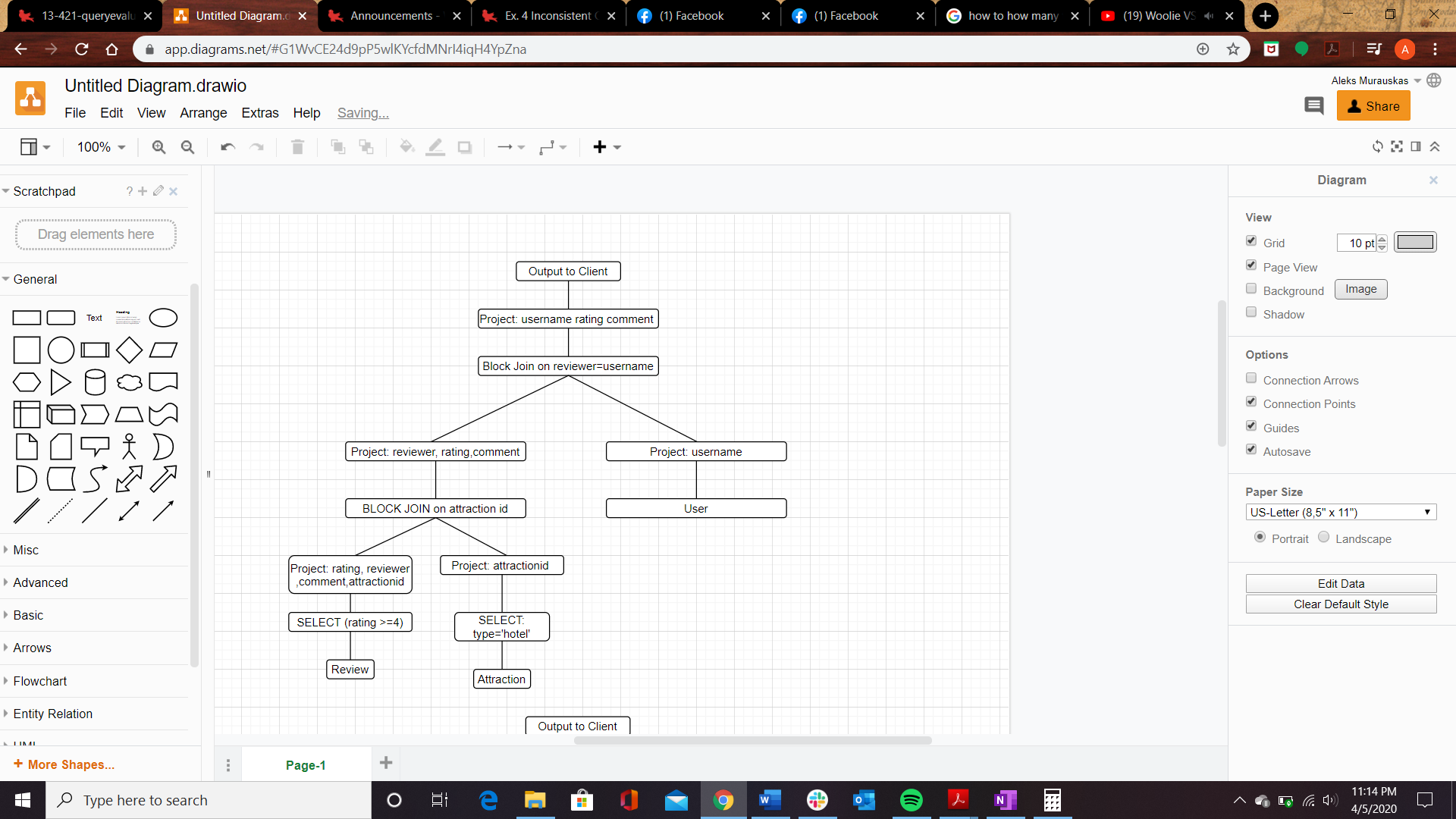
We can use block join easily

Now to join:

The final cost of this is

End Cost=

In Case 2:



We must project first so we can account for both

Review can be reduced: and only require rating, reviewer, comment and attractionid, size 253\*12,000/4000 =759 pages

If we write this to disk as temp t, we incur 759 cost

If we use this new location

Pre cost 759, COST to block join =

Attraction is reduced to 750 tuples and only attraction id is needed, 750\* 8 can easily be held in memory.

We begin by joining Review and attraction

Compare block joins

Block Join

Since there a reduction factor on both inputs we must project before,

With review projected:

Between these two, temporary T as the inner in a block join is preferred with the attractionids projected into main memory. 2 is preferred. Incurring a cost of 1743 IO.

The resulting tuples are 30000\* 1/10\* 2/5= 1200 tuples.

From this we project reviewer, rating, comment

1200\* (12+8+3/4(300) =1200\*245 = 294000 bytes, or 74 pages, this cannot be stored in main memory so it is written to disk as TEMP2, incurring 74 IO.

Lastly we join TEMP2 and User on username=reviewer.

If we project just username from User, we have 10000 entries of 12 bytes, or 30 pages. These 30 can held in main memory.

The final cost of case 2:

Case 2 is much more efficient

Ex. 4

SELECT R.attractionid, COUNT(DISTINCT R.reviewer)

FROM Attraction A, Review R

WHERE A.attractionid=R.attractionid

AND rating=1 AND A.type = `restaurant'

GROUP BY R.attractionid

Find the optimal execution plan and its cost (you need not draw the tree but it may help you find the solution faster). Assume that both Attraction and Review have unclustered type II indexes on their primary key. Assume no other indexes than these.

We are selecting 1/10th of attractions: therefore

We are selecting 1/5th of reviews: therefore

Number of reviews per attraction:

First we project

Attractionid and type from attraction: new size =8+8 =16. 750 tuples \* 16 = 12000 bytes or 3 pages

We project reviewer, rating, attractionid: new size = 12+8+8=28. 6000 tuples\*28 =168000 bytes. 42 tuples

Both of these can fit in main memory together.

See which is optimal Index Nested loop Join, or a Block nested join to determine which is less expensive

Check if index is better than block

Index is better if:

Block Nested Join

COST:

Case 1: Attraction is outer

COST:

Case 2: Review is outer

COST:

Case 1 is stronger, lets see if its stronger than Index nested join

Index Nested Loops Join: Inner is attraction

Cost of finding Inner tuple if inner is attraction: 1 leaf and 1 data pages

If inner is review

Inner is review:

Of our four options it seems block nested is the best option. Case 1

Number of tuples that are output of the join:

Tuples produced= 750\* 4= 3,000 tuples

However by the reduction factor in review, we have a total of 600 tuples

From this join attractionid and reviewer will be projected

size of a tuple will be 12+ 8=20 bytes. 20\*600 total bytes stored. 12000 bytes

12000/4000 =3 pages used. Will easily fit in main memory.

Next we perform an in memory sort on attraction id, then on Reviewer. This has no IO cost. We can do a pass over this sorted data to produce groups based on attractionid, keeping a counter for the distinct number of reviewers we have seen. This can be produced as the output of the group by. Again, everything is in memory, therefore no I/O cost.

Therefore the only I/O cost is associated with the join. Which shows the total cost of this query is 3115.