CS 486/586 Final

Fall 2021

Submitted By:Agnita Rayen

Instructions.

This file contains the Final Exam for CS 486-586 - Fall 2021.

To complete this exam, please make a copy of the google doc and edit it. When you are ready to turn in your exam, please save the doc as a pdf and turn it in on Canvas. (If you wish to use a different editing software, that is fine, what is required is that you turn in a nice, legible pdf.)

The midterm is **due Thursday 12/9 12:05 pm.**

This exam is **open book, open notes, open Internet, open everything**. However, if you use material from the Internet or sources other than the textbook, course slides or your notes, **you must cite the web site or material that you used.** For example, you could include a link to the web site you used or the name of a book you used in your answer.

200 points total.

Please do be sure to read questions completely.

Breakfast Schema with Catalog Tables:

This exam uses the Breakfast Schema, which was selected as you are somewhat familiar with it. I have added two catalog tables called relation_info and attribute_info, which provide information about relations and attributes. These tables are loosely based on the pg_class table and pg_stats table.

The underlined attributes are primary keys. There are indices on primary keys and on Companies.StockSymbol.

Breakfast Schema:

BreakfastFoods

Id	Name	Calories	Fat	Sodium	Carbs	MadeBy
1	Bacon, Sausage & Egg Wrap	640	33	1090	58	5
2	Scrambled Eggs (2 eggs)	149	11	145	1.6	--
3	Peanut Butter Homestyle Granola	140	7	65	18	3
4	Nonfat Greek Yogurt	90	0	65	5	4
5	Coffee with half and half	42	3.5	19	1.5	5

Consumers

Id	Name	FavoriteBreakfast
1	Olivia	3
2	Roman	--
3	Mikhail	4
4	Daniela	5
5	Miranda	5
6	Kiran	4

Notes:

- Consumers are people who eat breakfast.
- Calories, Fat, Sodium and Carbs are per serving.
- Bob's Red Mill and Fage are privately held and so do not have a stock symbol.
- Scrambled eggs are homemade, so 'Made by' is null

Sources:
- https://www.starbucks.com
- https://www.bobsredmill.com/
- https://fdc.nal.usda.gov
- https://usa.fage

Companies

Id	Name	StockSymbol
1	Hostess Brands, Inc	TWNK
2	J.M. Smucker Company	SJM
3	Bob's Red Mill	--
4	Fage	--
5	Starbucks	SBUX

relation_info

relationname	**numtuples**	**numpages**	**tuplesperpage**
BreakfastFoods	6,000	600	10
Consumers	100,000	5,000	20
Companies	80	4	20

relationname -- the name of the relation
numtuples -- is the number of tuples in the relation
numpages -- is the number of pages in the relation
tuplesperpage -- the average number of tuples on each page

attribute_info

relationname	**attrname**	**minvalue**	**maxvalue**	**distinctvalues**
BreakfastFoods	Id	1	200	200
BreakfastFoods	Name	--	--	200
BreakfastFoods	Calories	5	500	180
BreakfastFoods	Fat	0	15	16
BreakfastFoods	Sodium	0	1500	1000
BreakfastFoods	Carbs	0	40	30
BreakfastFoods	MadeBy	1	50	45
Consumers	Id	1	100,000	100,000
Consumers	Name	--	--	98,000
Consumers	FavoriteBreakfast	1	200	175
Companies	Id	1	40	40
Companies	Name	--	--	40
Companies	StockSymbol	--	--	40

relationname -- the name of the relation
attrname -- the name of the attribute
minvalue -- the smallest value that occurs in the table relationname for attribute attrname.
For example, the smallest number of Calories in any breakfast food in the BreakfastFood table is 5.
maxvalue -- the same as minvalue, but maximum value instead of minimum value
distinctvalues -- the number of distinct values in that attribute. For example, there are 200 different breakfast food Names in the BreakfastFood table.

Note: min and max values are not provided for string attributes

Name:

Q1 Index & Index Matching (15 points).
Below is a list of queries and a list of indexes. For each query, complete the table below to list the index or indexes that could be used to improve each query's performance and a brief justification — a few words or a short phrase — for choosing or not choosing an index. Consider both if the index matches the query and if the index will improve the query's performance.

Queries:

```
1. SELECT *
   FROM BreakfastFoods
   WHERE Id = 5;

2. SELECT *
   FROM BreakfastFoods
   WHERE Sodium <= 5 and Calories > 30;

3. SELECT *
   FROM BreakfastFoods B, Consumers C
   WHERE B.Id = C.FavoriteBreakfast
         AND B.Calories = 130;
```

Indexes:

A. clustered index on BreakfastFoods.Id
B. unclustered index on BreakfastFoods.Calories
C. unclustered index on BreakfastFoods.Sodium
D. clustered index on Consumers.FavoriteBreakfast

Query	Indexes	Brief Justification
1 (5 points)	A	Clustered index will give better performance on Primary keys as well as when we select all the columns for a given primary key value. With a clustered index the rows are stored physically on the disk in the same order as the index and it uses SEQ

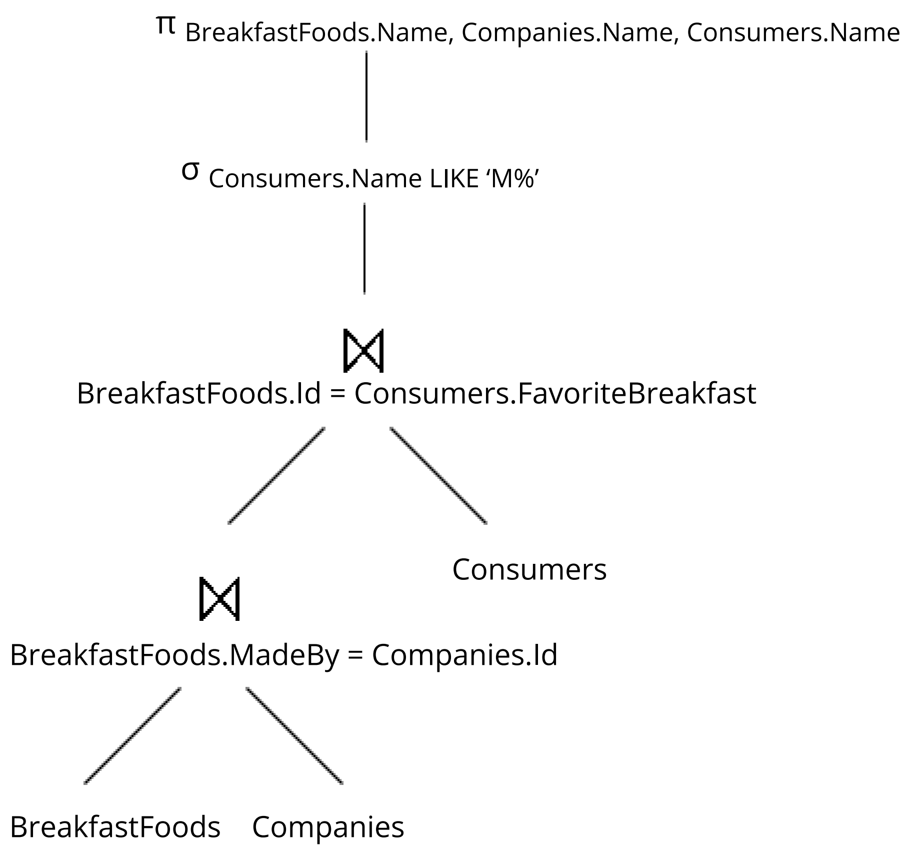
		SCAN. Here is the query analyzer Seq Scan on breakfastfoods (cost=0.00..1.06 rows=1 width=55) (actual time=0.021..0.022 rows=1 loops=1) Filter: (id = 5) Rows Removed by Filter: 4 Planning Time: 0.406 ms Execution Time: 0.072 ms But with the un clustered index, it would give better performance when you select the “indexed” column as it maintains a second list that has pointers to the physical rows.
2 (5 points)	B and C	Based on the conditions, we could apply a non clustered index. It is important to note that the clustered index doesn’t work with more than one condition. Non clustered index stores the column values in sorted order and can be applied with more than one. So calories and sodium are sorted indexes and their row addresses are fetched.
3 (5 points)	D and A	Since breakfast has a clustered index on the “id” column.If consumers also have an index on the join key column. Then join operation will be quicker instead of searching the entire table.

Q2 RA Equivalences (20 points).
Consider the Relational Algebra tree (a.k.a. logical query tree) below. Transform this tree twice - that is: transform the RA tree below into a new RA tree and then transform the new RA tree into a third RA tree.

- Use the two RA Equivalences below in your transformations.
- Use the RA equivalences in the order given, that is the first transformation should be done with RA 1 and the second transformation done with RA 2.

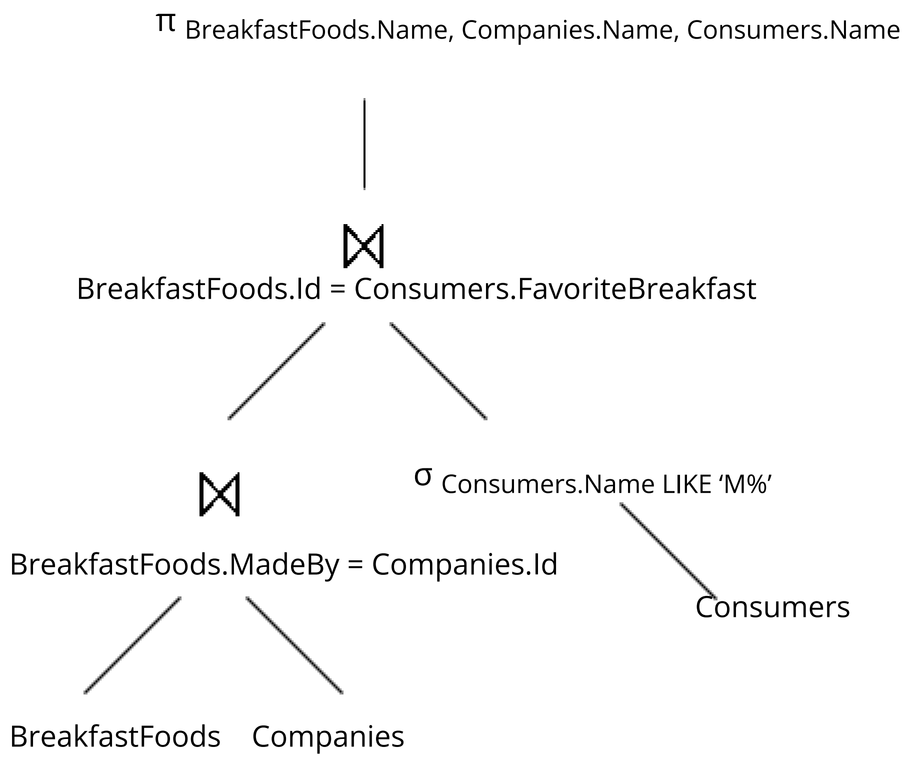
RA 1: $\sigma_c(R \bowtie S) \equiv R \bowtie \sigma_c(S)$

RA 2: $R \bowtie (S \bowtie T) \equiv (R \bowtie S) \bowtie T$.

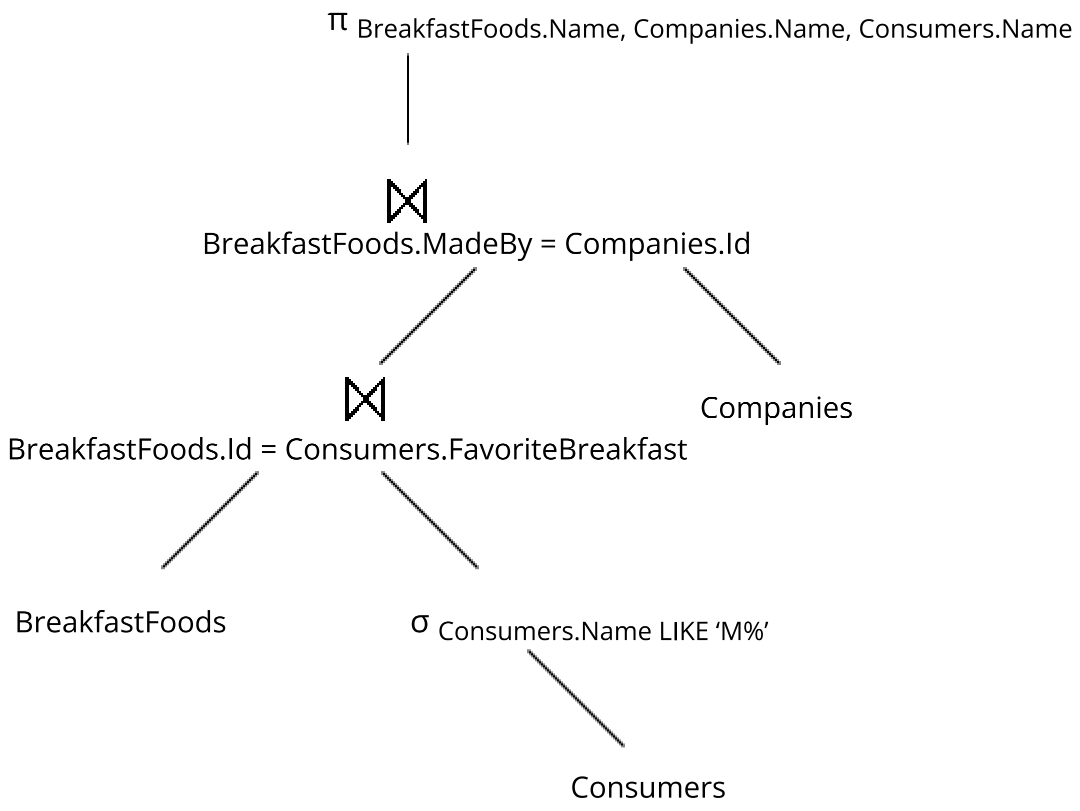

π BreakfastFoods.Name, Companies.Name, Consumers.Name
σ Consumers.Name LIKE ‘M%’
⋈
BreakfastFoods.Id = Consumers.FavoriteBreakfast
Consumers
⋈
BreakfastFoods.MadeBy = Companies.Id
BreakfastFoods
Companies

Solution:

RA 1: $\sigma_c(R \bowtie S) \equiv R \bowtie \sigma_c(S)$


π BreakfastFoods.Name, Companies.Name, Consumers.Name
⋈ BreakfastFoods.Id = Consumers.FavoriteBreakfast
⋈ BreakfastFoods.MadeBy = Companies.Id
σ Consumers.Name LIKE ‘M%’
Consumers
BreakfastFoods
Companies

RA 2: R⋈(S⋈T) ≡ (R⋈S)⋈T.


π BreakfastFoods.Name, Companies.Name, Consumers.Name
⋈ BreakfastFoods.MadeBy = Companies.Id
⋈ BreakfastFoods.Id = Consumers.FavoriteBreakfast
Companies
BreakfastFoods
σ Consumers.Name LIKE ‘M%’
Consumers

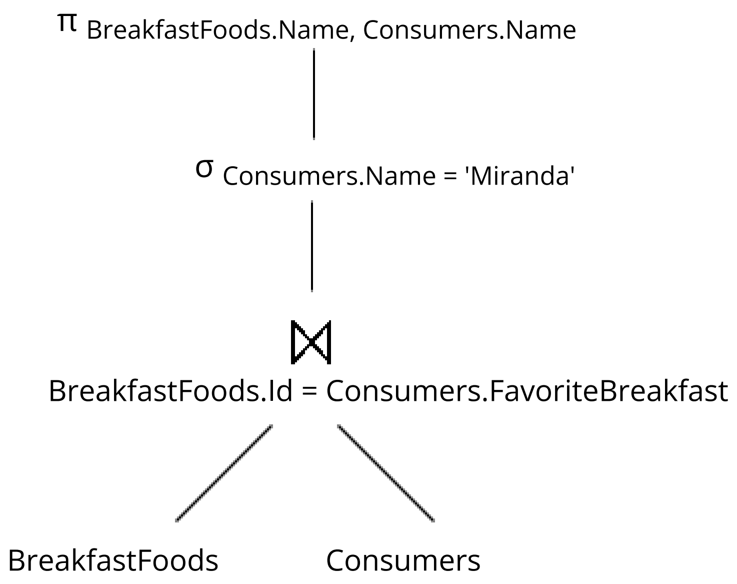
Q3 Cost Estimation (25 points).
For the two RA trees below, estimate the join cost assuming the query optimizer chooses a hash join.

For each RA tree, please:

- give the estimated cost of the join
- state which relation on which you would build the hash table
- for b., describe how you estimated the size of the right input to the join

Assume that the number of buffer pages available to the join is 75; that is BP = 75 (BP is the same as B). Please use the numbers in the relation_info table for your calculations.

a.


π BreakfastFoods.Name, Consumers.Name
σ Consumers.Name = 'Miranda'
⋈
BreakfastFoods.Id = Consumers.FavoriteBreakfast
BreakfastFoods
Consumers

Solution:

a.For each RA tree:

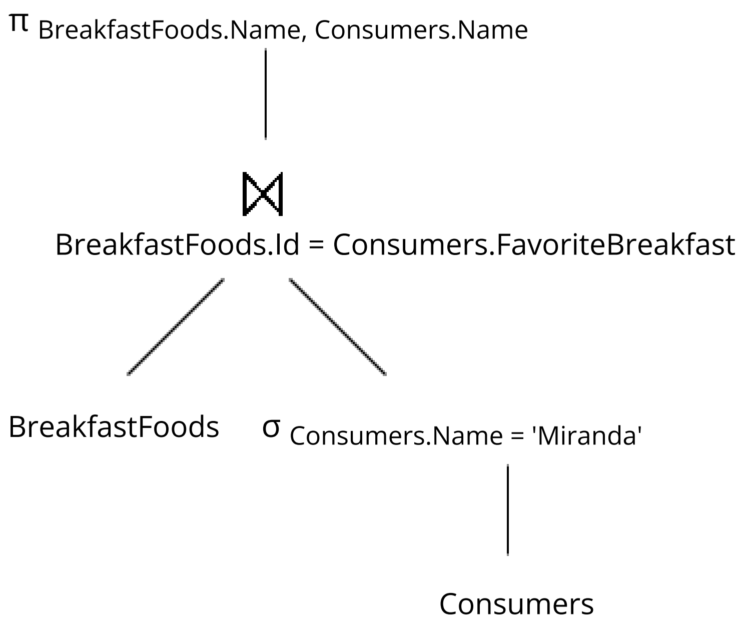
- give the estimated cost of the join

From the question,I'm assuming that it is a HASH JOIN.
Data from **relation_info**
No of page for BreakfastFood = 600
No of page for Consumers = 5000
BP=75
Then
One relation fits in the memory.So the hash join will be a simple case.
Estimated Cost of the join = M + N
= 600 + 5000 =5600
So the estimated cost of the join is 5600.

- state which relation on which you would build the hash table

  The hash condition from the above-given example tree would be "BreakfastFood.id = Consumers.FavoriteBreakfast".
  BreakfastFood table is much smaller than the Consumers table. So it fits into memory. Hence hash join is a more efficient solution.

b.


π BreakfastFoods.Name, Consumers.Name
⋈
BreakfastFoods.Id = Consumers.FavoriteBreakfast
BreakfastFoods
σ Consumers.Name = 'Miranda'
Consumers

Solution:

b.For each RA tree:

- give the estimated cost of the join

Again from the question,I'm assuming that it is a HASH JOIN.

Data from **relation_info, attribute_info**

No of page for BreakfastFood table = 600

No of pages with name "Miranda" = Total tuple - distinct tuple

= 100,000 - 98,000 = 2000

No of pages = Number of tuples / tuple per page

= 2000 / 20 = 100

Estimated Cost of the join = M + N

= 600 + 100 =700

Given BP is 75

Here one relation fits in the memory.So the hash join will be a simple case.

- state which relation on which you would build the hash table

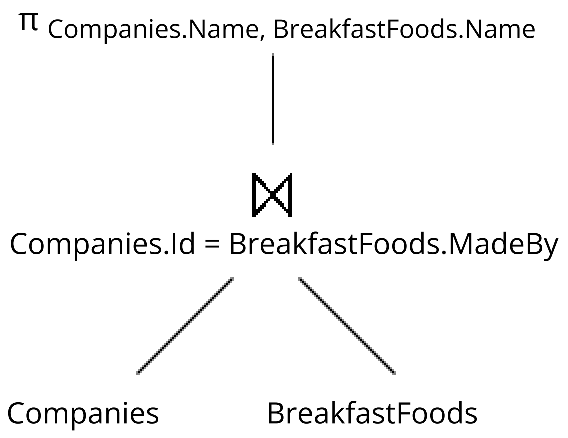
  The hash condition from the above-given example tree would be "BreakfastFood.id = Consumers.FavoriteBreakfast".
  Consumer table is much smaller than the BreakfastFoods .So it fits into memory. Hence hash join is a more efficient solution.

- for b., describe how you estimated the size of the right input to the join
  We need to check the condition first. Which are Consumers.Name ='Miranda'.Because the condition is based on name and name is only one

'Miranda'.I'm applying the condition in the below calculation. Given that is 98000 distinct value. So the rest of the value, I'm assuming is (100,000-98,000=2000) Miranda is 2000. The calculate the page using the formula is the number of tuples / tuples per page =2000/20=100.

Q4 Join Implementation (30 points).

Given the RA tree below and the hash function h(x) = x mod 4.


π Companies.Name, BreakfastFoods.Name
⋈
Companies.Id = BreakfastFoods.MadeBy
Companies
BreakfastFoods

a. Please draw the hash table that would be created for this join using the tuples given in Breakfast Schema.

b. Using the first tuple from the relation on which you did not build the hash table, please give a 3-step example of probing the hash table using that tuple.

Solution:

a.Please draw the hash table that would be created for this join using the tuples given in Breakfast Schema.

Given the hash function h(x) = x mod 4
Given join attribute is **companies.id**
For the companies.id hash values are below

Companies.id =1 -> 1 mod 4 =1
Companies.id =2 -> 2 mod 4 =2
Companies.id =3 -> 3 mod 4 =3

Companies.id =4 -> 4 mod 4 =0
Companies.id =5 -> 5 mod 4 =1
Using the hash value from above calculation the following hash table can be built on companies.

To build the hash table,we need to follow the following mentioned steps.
Step 1

- Build Hash Table on companies
- Take join att (companies.id)
- hashval =hash function h(x) =x mod 4 insert tuple into hash table based on the hash value.

Hash Value	Tuple
1	(1,'Hostess Brands, Inc',TWNK),(5,'Starbucks',SBUX)
2	(2,'J.M. Smucker Company',SJM)
3	(3,'Bob's Red Mill')
0	(4,'Fage')

b.Using the first tuple from the relation on which you did not build the hash table, please give a 3-step example of probing the hash table using that tuple.

We didn't use the breakfastFood for the above question (which is a) for the hash table.So probing the first table tuple from this relation in the hash table follows the following

- Scan BreakfastFoods for each tuple in BreakfastFoods,probe hash table on companies.
- take join attr(BreakfastFoods.madeby)
- hashval=hash fcn h(x) = x mod 4 for (breakfastFood)
- probe hash table using hashval

Q5 Query optimization - join selection (30 points)

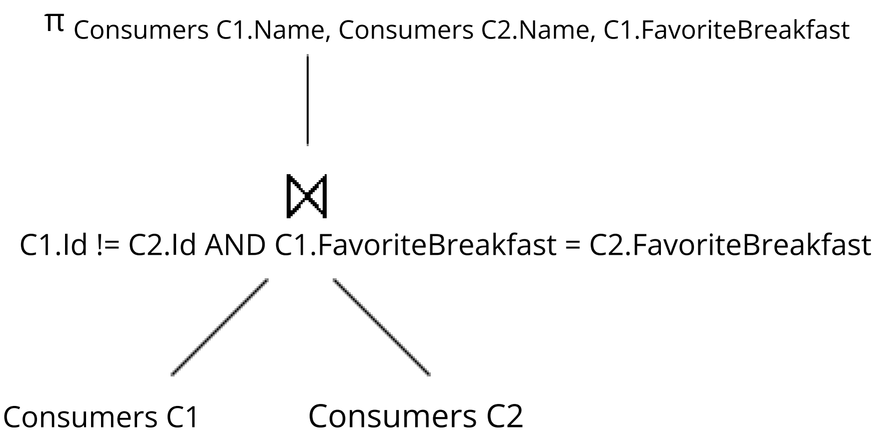
For the queries below, estimate the join costs. Assume BP = 75 (BP is same as B).
Please use the numbers in the relation_info table for your calculations.

a. Estimate the cost of this query using:

i. HashJoin

ii. Sort-Merge Join

iii. BlockNestedLoop Join


π Consumers C1.Name, Consumers C2.Name, C1.FavoriteBreakfast
⋈
C1.Id != C2.Id AND C1.FavoriteBreakfast = C2.FavoriteBreakfast
Consumers C1
Consumers C2

Solution:

a.Estimate the cost of this query using:

i. HashJoin

No of pages for consumers C1 =5000

No of pages for consumers C2 = 5000

Cost of Hash Join = C * (M+N) = C * (5000+5000) = C * (10,000) where C > 3.

ii. Sort-Merge Join

For this case

No of pages for consumers C1 =5000

No of pages for consumers C2 = 5000

Cost of sort merge join = C * (M+N) = C * (5000+5000) = C * (10,000) where C > 3.

iii. BlockNestedLoop Join

For this case

No of pages for consumers C1 =5000

No of pages for consumers C2 = 5000

Cost of BlockNestedLoop join = M + (M/(BP-2)) * N

= 5000 + (5000/(75-2)) * 5000

= 5000 + (68.49) * 5000

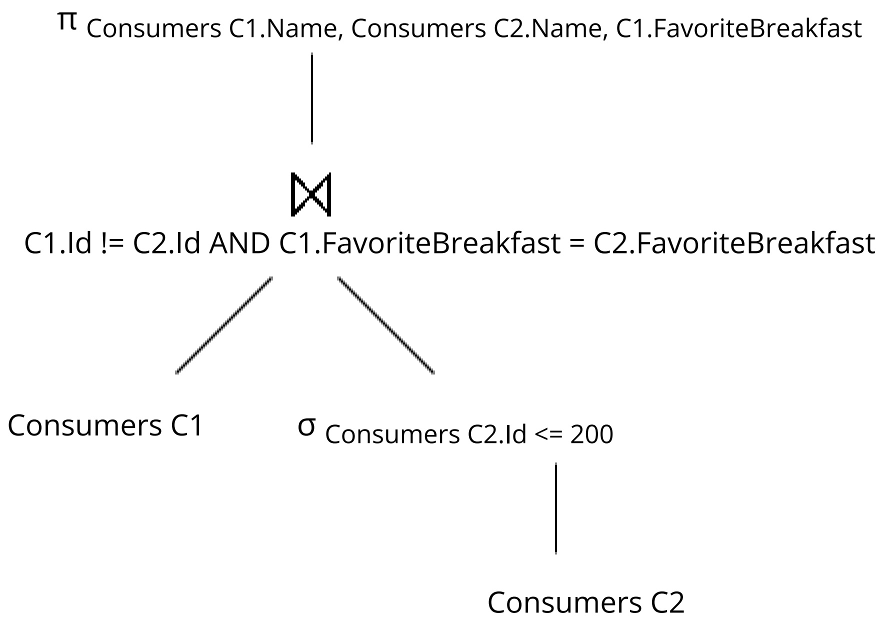
= 347,465.753

b. Estimate the cost of this query using:

i. HashJoin

ii. Sort-Merge Join

iii. BlockNestedLoop Join


π Consumers C1.Name, Consumers C2.Name, C1.FavoriteBreakfast
⋈
C1.Id != C2.Id AND C1.FavoriteBreakfast = C2.FavoriteBreakfast
Consumers C1
σ Consumers C2.Id <= 200
Consumers C2

Solution:

b.Estimate the cost of this query using:

i.HashJoin

No of pages for consumers C1 =5000

No of pages for consumers C2 = 200 tuple based on condition according to relation table 20 per page.so we need 10.

So no of pages for table consumers C2 =10

Then

Hash join = M + N = 5000 + 10 =5010

ii.Sort-Merge Join

No of pages for consumers C1 =5000

No of pages for consumers C2 = 200 tuple based on condition according to relation table 20 per page.so we need 10.

So no of pages for table consumers C2 =10

Then

sort-merge join = 3 * (M + N) = 3 * (5000 + 10) = 15,036

iii.BlockNestedLoop Join

No of pages for consumers C1 =5000

No of pages for consumers C2 = 200 tuple based on condition according to relation table 20 per page.so we need 10.

So no of pages for table consumers C2 =10

Then

BlockNestedLoop Join = M + (M /(BP-2)) * N

= 5000 + (5000/(75-2)) * 10

=5684.93151

c. For the hash joins, what attribute would you build the hash join on?

Solution:
For hash join,
we will be build the hash on join attribute
C1.FavoriteBreakfast = C2.FavoriteBreakfast
In this joint relationship where ids are not equal.

Q6 Query Optimization (20 points).
For the query in Q5.b., give two specific examples of how your estimate of the cost of the join could be incorrect. (Please do not just repeat the information in the slides, instead, translate that information into specific examples using the query above.)

Solution:
We consider all 200 tuples as values in the cost calculation.If we do not have 200 tuples after applying the conditions and we have only get 100 tuples after applying the condition.Then number of pages required for this 100 tuples is just 5 instead 10.This will change the calculation for each join.

For instance:

Hash Join :
No of pages for consumers C1 =5000
No of pages for consumers C2 = 5
= M + N = 5000 + 5 = 5005

Sort-Merge join:
No of pages for consumers C1 =5000
No of pages for consumers C2 = 5
= 3* (M + N) = 3*(5000 + 5)= 15015

BlockNestedLoop Join:
No of pages for consumers C1 =5000
No of pages for consumers C2 = 5

BP =75

= M + (M /(BP -2)) * N

= 5000 +(5000/(75-2) * 5 = 5342.4657

My next approach is
Q5 b might be wrong when catalog stats may be out of date.
Assume that an additional 30000 tuples have been added to the table consumers so that makes the total tuples 140,000 instead of 100000.Now for 140000 tuples,database needs 7000 pages for this total data that is based on 20 tuples per page.If database checks the catalog to identify the least cost of join before catalog is updated so M changes from 5000 to 7000.Data base may select the wrong join in this estimation.Cost itself would be wrong.

When M =5000
Hash join = M + N = 5000 + 10 =5010

When M =7000
Hash join = M + N = 7000 + 10 =7010

Similarly
sort-merge join = 3 * (M + N) = 3 * (5000 + 10) = 15,036
sort-merge join = 3 * (M + N) = 3 * (7000 + 10) = 21,030

BlockNestedLoop Join = M + (M /(BP-2)) * N
= 5000 + (5000/(75-2)) * 10
=5684.93151

BlockNestedLoop Join = M + (M /(BP-2)) * N
= 7000 + (7000/(75-2)) * 10
=7958.9

When databases use incorrect values such as out of date datas,it might give incorrect results.

Q7 Concurrency (15 points).
Sometimes concurrency causes issues, sometimes it does not cause issues. For each pair of queries below tell whether there are any potential issues if that pair of

queries ran at the same time or if there are no potential conflicts between the two queries.

Q1: UPDATE BreakfastFoods SET Calories = 210 WHERE Id = 1;
Q2: SELECT * FROM BreakfastFoods WHERE Id = 1;
Q3: UPDATE Companies SET Name = 'Peets' where Id = 5;

Q1 & Q2:
Q1 & Q3:
Q2 & Q3:

Solution:
Q1 & Q2:
Q1: UPDATE BreakfastFoods SET Calories = 210 WHERE Id = 1;
Q2: SELECT * FROM BreakfastFoods WHERE Id = 1;

If we run Q1 and Q2 at the sametime.There will be a conflict .Because we are performing both the tasks on the same table and also on the same tuple like id =5.So if we generate at the sametime we might not get the accurate output from selected query.It could give either 640 or 210.

Q1 & Q3:
Q1: UPDATE BreakfastFoods SET Calories = 210 WHERE Id = 1;
Q3: UPDATE Companies SET Name = 'Peets' where Id = 5;

If we run Q1 and Q3 at the sametime nothing will happen.Because both are different table.so it won't create any issues.

Q2 & Q3:
Q2: SELECT * FROM BreakfastFoods WHERE Id = 1;
Q3: UPDATE Companies SET Name = 'Peets' where Id = 5;

If we run Q2 and Q3 together, it won't create any issues. Because both are different table.Even though Q2 and Q3 used ids. The ids are the same but the name only changes in Q3.So it won't affect the table.

Q8 Concurrency (10 points).
List two reasons supporting concurrency is a good idea.

Reduced waiting time

Concurrent execution of transactions would reduce the waiting time of other transactions. Consider a situation where all transactions are executed serially. Transactions may be of any size. When executing serially, one long transaction is executing and a very small transaction may be on the queue for its turn. This increases the waiting time of small transactions. Hence, if we execute transactions simultaneously, the waiting time would be much reduced when compared to the serial execution.

Average response time of transaction increased

The average time consumed by a transaction to complete since its start is called the average response time. In concurrent execution, as multiple transactions are executing simultaneously by sharing the system resources, the waiting time is reduced which in turn increases the average response time.

Q9 Recovery (20 points).
Peanut Butter Granola has become a very popular breakfast food. Consider the series of SQL commands below which modify the database to show that Daniela, Miranda and Kiran all now list Apple Blueberry Granola as their favorite breakfast food.

```
BEGIN;
UPDATE Consumers SET FavoriteBreakfast = 3 WHERE Id = 4;
UPDATE Consumers SET FavoriteBreakfast = 3 WHERE Id = 5;
UPDATE Consumers SET FavoriteBreakfast = 3 WHERE Id = 6;
COMMIT;
```

At this point in time, after these commands have been executed:

- The page that contains the tuple with Id = 4 (Jasmine) is in memory.
- The page that contains the tuple with Id = 5 (Miranda) has been written to disk.
- The page that contains the tuple with Id = 6 (Jorge) has been written to disk.

And now, the system crashes and memory is lost.

Please answer:

a. Which of the three tuples (give Id and Name) are problematic in this situation?

b. What needs to be done to that tuple or tuple to fix the issue? Redo or Undo?

Solution:

a.Which of the three tuples (give Id and Name) are problematic in this situation?

From the given 3 tuples,the page that contains the tuple with id=4 name:Daniela is problematic.It has been in memory and we have committed the query. Now the system crashes and memory lost.
The pages that contain the tuple with id=5 and id=6 have already been written to the disk and committed. so we are not going to have any problem with these tuples.

b.What needs to be done to that tuple or tuple to fix the issue? Redo or Undo?
We made an update and it has not been written to disk, updates are in memory also transaction has been done but the system crashes. So "write to

late" is REDO.To fix the tuple with id=4 as the tuple is committed but still in memory.In this   case we need to do redo.

Q10 Transactions & Recovery (15 points).
You receive a tip that Olivia's (Id 1) favorite breakfast is now Coffee with half & half, you begin to write a query to update the database... but then you are told that actually, Olivia's favorite breakfast has not changed and remains Peanut Butter Granola, thank goodness you got that information before you'd committed the transaction. Thus, you have typed the following series of SQL commands.

```
BEGIN;
UPDATE Consumers SET FavoriteBreakfast = 5 WHERE Id = 1;
ROLLBACK;
```

At this point, the page containing the tuple with Id=1 has been written to disk.

Please answer:

a. Do you have a problem or not?
b. If you do have an issue, what needs to be done to fix the issue? Redo or Undo?

Solution:

a.Do you have a problem or not?
No,I don't have any problem.Because in this translation, we have made an update query, and then the update has been written to disk. Then we have used ROLLBACK.DBMS uses the log to restore the previous values even if the item was written to disk. So there is no problem with this set of questions.

b.If you do have an issue, what needs to be done to fix the issue? Redo or Undo?
Here we made an update then the update was written to disk but the transaction was not committed but the system crashes.It is written too early so it's UNDO.

In this case I’m assuming there is an issue.We need to perform UNDO.Because the values were already written to the disk and indo is the only option for us.