Solutions to PS3

Name : Vinay Ninganagouda Patil
NetId : vpatil3
Student Id : 9079431293

A : THE YELP DATABASE

A1.

Solution :

The different indexes on the table Users are that match the predicate in the SQL query are,

i) Hash Index on (Name)
ii) B+ Tree Index on (Age)
iii) B+ Tree Index on (Name, Age)
iv) B+ Tree Index on (Age, ReviewCount)
v) B+ Tree Index on (Name, ReviewCount)

Reasoning:

The simplified version of the selection condition looks like below,

(Name = ”John” OR Name = ”Mary”) AND (Age = 20 OR Age > 50)

i) The Hash Index on (Name) matches the selection conditions - Name = ”John”, Name = “Mary”
ii) The B+ Tree Index on (Age) matches the selection conditions - Age = 20, Age > 50
iii) The B+ Tree Index on (Name, Age) matches the selection conditions - Name = ”John”, Name = “Mary”, because of the prefix match
iv) B+ Tree Index on (Age, ReviewCount) matches the selection conditions - Age = 20, Age > 50, because of the prefix match
v) B+ Tree Index on (Name, ReviewCount) matches the selection conditions - Name = ”John”, Name = “Mary” , because of the prefix match

---

A2.

Solution :

Calculation of I/O Costs,
Note that the cost of writing the output of the join is excluded

(a) ***Block Nested Loop Join***:
Since it’s a Naive Block Nested Loop Join, it doesn’t take advantage of the B+ tree index on the join attributes.

It's I/O cost is given by, $M_R + M_S$. ceil (M_R / (B-2))
Here let M_R = Number of Pages in Checkins = 20,000,
M_S = Number of Pages in Businesses = 42,000,
B = Number of Buffer Pages = 10,000

Now the **I/O cost** will be = 20,000 + 42,000 . ceil(20,000/(10,000-2)) = **104,000**

(b) ***Sort-Merge Join***:
Since B+ tree index is used on both joining indexes of the relations, and both indexes follow the alternative of storing the data records directly in the leaf pages of the index, they are sorted. So we can skip the first step which involves sorting the relations,

And since both the join attributes are primary keys in the relations, there won't be duplicates, which means no backup

So the total I/O cost is equal to I/O cost of read + merge = $M_R + M_S$
Here let M_R = Number of Pages in Checkins = 20,000,
M_S = Number of Pages in Businesses = 42,000

Now the **I/O cost** will be = 20,000 + 42,000 = **62,000**

(c) ***Hash Join***:
We should note that, the sorting of data doesn't provide any improvements to the Hash Join algorithm
Lets create k = B-1 partitions in one pass = 10,000 - 1
Here let M_R = Number of Pages in Checkins = 20,000,
M_S = Number of Pages in Businesses = 42,000
And since, B^2 = 10,0000000 is greater than M_R (Which is the minimum no of pages in the two relations), I/O cost for Hash Join algorithm will be $3(M_R + M_S)$

Now the **I/O cost** will be = 3 * (20,000 + 42,000) = **186,000**

Hence looking at the calculations above we can say that the **Sort-Merge Join has the lower I/O cost** for a natural join of Businesses and Checkins.

A3.

Solution :

We have B-1 buffer pages left for the hash table (Considering 1 page for input).

Here let B = Number of Buffer Pages = 10,000

Given Page size = 8kB = 8 * 1024 Bytes

Hence, the max space available to construct hash table is given by,

(B-1)*8kB = (10,000-1)*8*1024 = **81,911,808 Bytes**

Now, we know that for hash based aggregation, each entry of the hash table will represent one group, and each entry will contain City – **20 Bytes** and Count(BusinessID) – **8 Bytes** (Integer).

Hence the total size of each entry is **28 Bytes.**

Now looking at these details we can say that the maximum number of cities that can fit is equivalent to the no of entries in the hash table.

And we also know that the size of hash table cannot exceed **(B-1)*8kB**,

So, with Cities = No of hash table entries,
We can say, f*Cities*28 <= 81,911,808
Which gives us Cities <= **2,089,586.93**

Hence we can say that, the maximum number of cities for which it is possible to implement hash-based aggregation using a one pass algorithm is **2,089,586**

A4.

Solution :

For simplification of explanation, lets define some variables beforehand,
R = Users,
M_R = Number of Pages in Users = 75,000,
S = Reviews,
M_S = Number of Pages in Reviews = 500,000
(Note that the cost of writing the output of the join is excluded)

The question clearly states that the values of Stars and Age are uniformly distributed. Which means that a finite number of values are equally likely to be observed.

Also since the range of Age is large and the no of pages in Users relation is smaller when compared to Reviews and the fact that those 2 factors influence selectivity . We want to choose the most selective, so I propose the following physical plan,

Let's calculate the selectivity of Users relation for U.Age = 18,

Selectivity = 1/Range of Age = 1/(99-10 + 1) = 1/90 = 1.11%

And since no indexes are available on any relation, we will resort to scan all pages of Users table and apply select condition of Age=18.

This gives an I/O cost of M_R = 75,000.

Let's approximate the max no of records the Users relation holds,
Size of each record of Users Relation = 8+30+8+8 Bytes = 54 Bytes

=> No of records per page = 8*1024 / 54 = 151

=> Max no of records in Users relation = Rec = 151 * 75,000 = 11,325,000

Since the selectivity is 1.11%, we can say that approximately Rec * 1.1% records were selected,

=> Records selected = Rec * 1.1 % = 11,325,000 * 1.1/100 = 125,575 (Approximately)

=> No of pages allocated for Users relation in buffer pool = 125,575/ 151 = 831 (Approximately)
(Note that, this could have also been calculated using selectivity on no.of pages directly, but I have instead used on no.of. records for more accuracy)

Since, 831 is << than the total no of available buffer pool pages, we can retain these pages in memory.

Also, lets assign a buffer page to hold the count value, which will be initialized to zero first.

Now let scan each page of Reviews one by one and for each page of Users relation(Already available in Buffer) compare to see if the value of Stars in Reviews record is less than 2 and if this true, then match UserId's of both the relation records, if both the conditions hold, then increment the counter we maintain in a buffer page. Then scan the next page and the cycle continues.

(Note: We could have also applied selectivity on Reviews relation instead of getting every page in the buffer space. Though this would have reduced the amount of memory used and reduced the no of comparison in the last step, but it wouldn't have influenced the I/O cost!)

As we can see this has been basically reduced to BNLJ, where the smaller relation fits in the memory (Buffer space). Hence the total I/O cost will be ,

$M_R + M_S = 75{,}000 + 500{,}000 = 575{,}000$
Hence, we have constructed a query plan, where the I/O is the cost of only scanning each relation once, which has to be the smallest possible I/O cost.

A5.

Solution :

$$R_1 \leftarrow \pi_{UserID}(\sigma_{ReviewCount>=100} Users)$$

$$R_2 \leftarrow \pi_{UserID,BusinessID}(\sigma_{Stars>4} Reviews)$$

$$R_3 \leftarrow R_1 \bowtie R_2$$

$$R_4 \leftarrow \pi_{BusinessID}(R_3)$$

$$R_5 \leftarrow \pi_{BusinessID,BName}(Businesses)$$

$$R_6 \leftarrow \pi_{BName}(R_4 \bowtie R_5)$$

B: ADDITIONAL QUESTIONS

B1.

Solution :

Given:

No of pages in S = $4BN_s$

No of pages in R = $8BN_R$

$4BN_s >> 8BN_R$

No of Buffer Pages = 4B + 1

Also, 2 f $N_R = 4B - 1$

Now for the initial partition of S, we allocation 1 buffer page to hold the read page of S and 4B buffer pages to create 4B partitions using hash function h.

After first hash partition of S, we get 4B partitions each of length N_s pages

=> Now to do this partitioning we perform one pass on the complete relation S and write the result back to the disk in the form of partitions.

=> I/O cost = $\mathbf{4BN_s + 4BN_s = 8BN_s}$ ---------- **(1)**

Now, similarly after the first partition of R, we get 2B partitions of length N_R pages, B partitions of length $2N_R$ pages, and B partitions of length $4N_R$ pages.

=> Now to do this partitioning we perform one pass on the complete relation R and write the result back to the disk in the form of partitions.

=> I/O cost = $\mathbf{8BN_R + 8BN_R = 16BN_R}$ ---------- **(2)**

Now, after this partitioning process, we have 4B partitions of S and R respectively in the disk.

Let's also note that, from $4BN_s >> 8BN_R$ and 2 f $N_R = 4B - 1$,

We can say that, for f>=1, $\mathbf{2N_R <= 4B - 1}$ **, and** $\mathbf{N_s >> 2N_R}$ ---------- **(3)**

Coming to the Join phase, let's consider 3 cases,

Case 1: Consider 1st 2B partitions of S and 1st 2B partitions of R for BNLJ join

Now, the no of pages in S will be = $2B * N_S = 2BN_S$

And the no of pages in R will be = $2B * N_R = 2BN_R$

From (3), we can add that, the $2N_R$ pages of each partition of R considered here fit in the Buffer space with 2 Buffer pages to spare (1 is allotted for read page of S and 1 for the join operation)

Hence, the BNLJ can accomplish the join operation in just one pass, as the smaller relation (R) in this case fits in the memory.

So, the total I/O cost for this Join operation (Excluding the I/O to write join result to disk) is given by,

$2BN_S + 2BN_R$ ---------- (4)

Case 2: Consider next B partitions of S and next B partitions of R for BNLJ join

Now, the no of pages in S will be = $B * N_S = BN_S$

And the no of pages in R will be = $B * 2N_R = 2BN_R$

From (3), we can again add that, the $2N_R$ pages of each partition of R considered here fit in the Buffer space with 2 Buffer pages to spare (1 is allotted for read page of S and 1 for the join operation)

Hence, the BNLJ can again accomplish the join operation in just one pass, as the smaller relation (R) in this case fits in the memory.

So, the total I/O cost for this Join operation (Excluding the I/O to write join result to disk) is given by,

$BN_S + 2BN_R$ ---------- (5)

Case 3: Consider last B partitions of S and last B partitions of R for BNLJ join

Now, the no of pages in S will be = $B * N_S = BN_S$

And the no of pages in R will be = $B * 4N_R = 4BN_R$

From (3), we see that, the $4N_R$ pages of each partition of R considered here won't fit in the Buffer space.

Now, to fix this we resort to recursive repartitioning(Assuming perfect uniform splitting) of these pages of the partitions in consideration.

Now using hash function h1, we partition all the records in BN_S pages(Relation S) into 4B partitions, so the size of each partition now will be $BN_S / 4B = N_S/4$ pages.

And similarly using h1, we partition all the records in $4BN_R$ pages(Relation R) into 4B partitions, so the size of each partition now will be $4BN_R / 4B = N_R$ pages.

Now this repartitioning will cost us below I/O,

$2*BN_S + 2*4BN_R = 2BN_S + 8BN_R$ ---------- (6)

From (3), we can say that, the N_R pages of each partition of R considered here fit in the Buffer space with 2 Buffer pages to spare (1 is allotted for read page of S and 1 for the join operation)

Hence, the BNLJ can again accomplish the join operation in just one pass, as the smaller relation (R) in this case fits in the memory.

So, the total I/O cost for this Join operation (Excluding the I/O to write join result to disk) is given by,

$BN_S + 4BN_R$ ---------- (7)

After this step, now the whole join operation has completed.

Therefore, the total I/O cost of this operation is given by, (1), (2), (4), (5), (6), (7)

$= (8BN_S) + (16BN_R) + (2BN_S + 2BN_R) + (BN_S + 2BN_R) + (2BN_S + 8BN_R) + (BN_S + 4BN_R)$

$= 14BN_S + 32BN_R$

B2.

Solution :

Given:
No of pages in relation R = N = 1,000,000

Size of the Buffer Pool = B = 1,100

To compute mode of attribute A of the Relation, let create sorted runs of R.

To do that, let's create B-1 chunks of Relation R.

Now, each chunk has approximately, N/B-1 = 1000000/1099 = 910 pages

Now read each chunk into the buffer and perform in-place sort (Quick Sort) and write these runs back to the disk.

This performs one pass on the relation and does 1 read + 1 write operation.

Hence, the I/O cost will be **2N**

Now, to calculate the modes of attribute A,

We allocate 1 buffer page to maintain mode, we keep four variable within it to calculate mode,

RunningCount = 0,

RunningNumber = NULL,

ModeCount = 0,

ModeList = []

Now, we read pages one-by-one from each sorted B-1 runs into B-1 Buffer Pages (i.e. 1 Buffer Page for 1 Run to be read in order).

Then calculate minimum value of A from the pages currently in B-1 pages and delete that record from the page, then follow the below algorithm,

Step 1: Check if RunningNumber == current minimum value, then increment the RunningCount variable.

Step 2: If RunningNumber != current minimum value, go to Step 3

Step 3: If RunningCount ==ModeCount, goto Step 4, else go to Step 5

Step 4: Append RunningNumber to ModeList, set RunningNumber = current minimum value and RunningCount = 1

Step5: If RunningCount > ModeCount, set ModeList = [current minimum value], and ModeCount = RunningCount

As and when the buffer page is exhausted of records, read the next page from the runs in order. Continue until all pages of the runs are traversed in order.

This Process of calculation of Mode does 1 read operation of all pages of Relation R,

i.e. I/O = **N** and if we consider writing the mode buffer page back to disk, we perform additional **1** I/O.

Hence the total I/O required to compute the mode is 2N + N + 1 = **3N + 1**
