COL 362/632: Minor – II

March 26, 2019

NAME:

DIVYANSHU SAXENA

ROLL NO:

2016CS10316.

NOTE:

1. Please write only in the space provided. We are using the tool Gradescope which automatically determines where your answers are written. Anything outside the boxes will be disregarded.
2. Some questions require calculations. Expressions which lead to the answer are sufficient, the exact number(s) are not required.

1 Part A

1. (**1 point**) What factor(s) determine the *access time* in a magnetic disk? (*Note:* Only precise terminology will be accepted as correct.)

   The SEEK TIME to seek the requisite track and the ROTATIONAL LATENCY to get to the requisite sector determine the access time.

2. (**1 point**) Give an example of a tertiary storage medium.

   USB Sticks.

3. (**2 points**) Give an expression to compute the storage capacity of a magnetic disk. Precisely state the parameters on which the storage capacity depends.

   No. of platters in a cylinder. × No. of tracks in a platter × No. of sectors in a track × No. of Bytes in a sector

4. **(1 point)** Which table(s) in the PostgreSQL system catalog store(s) statistics about the database such as number of tuples in the table?

pg_table stores the statistics about the database.

5. **(2 point)** We studied several different page layout methods in class. Which method(s) are used by PostgreSQL to store tables? Draw the layout structure(s).

6. **(1 point)** Given a table $R(\underline{A}, B, C, D)$ what can you say (if anything) about the underlying organization of the table if it is known that the attribute A has a *dense, secondary* index built on it.

Because the index built on A is secondary, hence we can conclude that the file organisation is definitely not sorted according to A [because of the definition of secondary index]. Hence, the file of the table records is sorted as per some other key.

7. **(2 points)** Suppose you are given the following information about a B^+tree index: i) the height of the tree is h, ii) no. of search keys stored currently is k. What other information (if any) can you conclude about the structure of the index?

The minimum number of keys $= \left(\lceil \frac{n}{2} \rceil\right)^h \cdot \left(\lceil \frac{n-1}{2} \rceil\right)$.

Maximum number of keys $= n^h - 1$

Therefore; $\left(\lceil \frac{n}{2} \rceil\right)^h \left(\lceil \frac{n-1}{2} \rceil\right) \leq k \leq n^h - 1$ where n is the parameter of BTree we can calculate this fanout.

2 Part B

1. **(3 points)** Suppose you are given with the following query.

```
select id
from instructor
where dept_name = 'finance' and salary = 80000
```

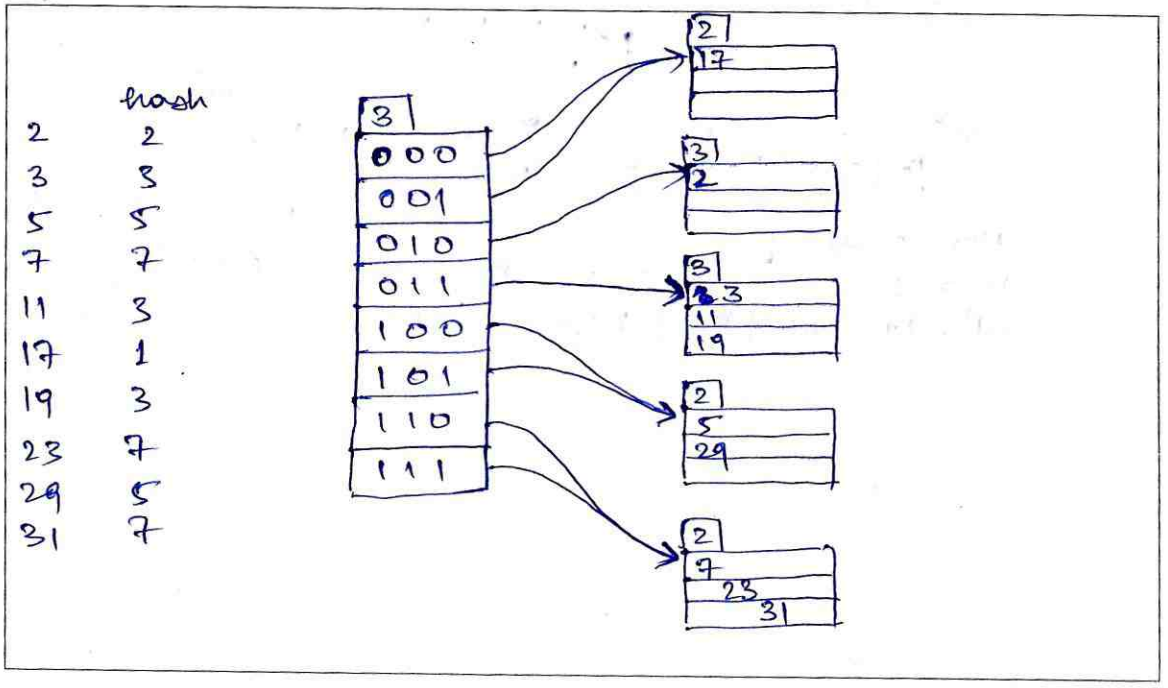
Give *three* different ways in which one or more indexes can be used to evaluate this query. Clearly state your assumptions about the indexes (if any).

(1) Use index over ~~department~~ dept_name to get all ~~instructor~~ records for which dept_name = 'finance'. Then search for the records with salary = 80000.

(2) Use index over salary to get all records for which salary = 80000 and then search for the records with dept_name = 'finance'.

(3) Use a composite multiple key index structure to find all records with ~~key~~ search-key = ('finance', 80000).

2. **(3 points)** Suppose you are given with the hash function $h(x) = x\%8$ (that is, the remainder when x is divided by 8). Show the hash index on inserting the following sequence of values: $2, 3, 5, 7, 11, 17, 19, 23, 29, 31$. Assume that the method of constructing the hash index is *extendible hashing* and that each bucket can store 3 values. (*Note:* Only the final structure needs to be shown. There will be no partial credit for this question.)



3. (**4 points**) Suppose we are required to construct a B^+*tree* index. Initially the tree is empty and we need to insert the key values one by one. Suppose this is done in the *sorted* order of the keys, what can be concluded about the occupancy of the leaf nodes? Construct a toy example to demonstrate your conclusion.

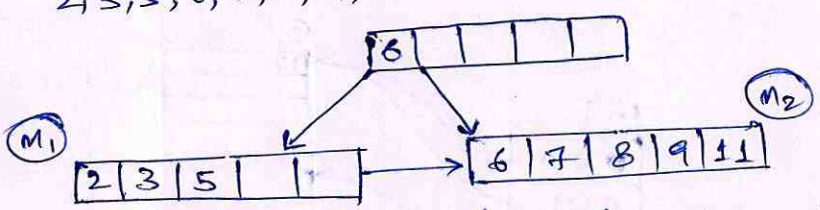
For the parameter n of a B^+ tree, all but one leaf nodes shall have $\lceil \frac{n-1}{2} \rceil$ keys (minimum allowed) and one leaf node shall have a maximum of $n-1$ keys.

~~That is, one leaf node shall be fully occupied and all other leaf nodes shall, one vacant key place have.~~

This happens because, during insertion the node is split into two nodes of half the maximum occupancy (which is equivalent to the minimum allowed occupancy). Now, because the insertion is taking place in sorted order, hence ~~there one~~ for one of the nodes no further insertion can take place and it shall remain at half occupancy.

Consider a B^+ tree with parameter $n=6$. And consider the following insertions:→

2, 3, 5, 6, 7, 8, 9, 11



Note that if any other insertion takes place, it shall be >11 and hence, no key can be added in leaf node (M1). It has $\lceil \frac{n-1}{2} \rceil$ keys (minimum allowed).

4. (**4 points**) Suppose we need to sort 256,000,000 numbers. Assume that each number occupies 4 bytes. The block size is $1024K$. Suppose the number of buffers available in memory are 16. Calculate the following: i) no. of sorted runs in the very first pass of the algorithm, ii) no. of merge passes required, iii) total no. of sorted runs produced, and, iv) size of the largest sorted runs (this occurs just before the final merge pass). State any other assumptions that you require clearly.

(i) $\frac{1024}{16}$

5. Recall the *nested loop* and *block nested loop* join algorithms that we studied in class. Suppose we need to join two relations R and S, but we need not store the result (that is, no buffer space is required to hold the results). Answer the following questions on the costs of these two algorithms. Consider only the block transfers in your costs.

   (a) (**3 points**) Compare the costs of the two algorithms, given that M buffers are available in memory and *neither relation fits entirely into memory*. Which of them is more efficient? State your assumptions clearly.

   (b) (**2 points**) Given that M buffers are available in memory and *neither relation fits entirely into memory*, are there any conditions under which the choice of algorithm does not matter? Explain your answer briefly.

5. The *division* operator in relational algebra, denoted by $\div$ is defined as follows. Let $r(R)$ and $s(S)$ be the relations with attribute sets R and S respectively. Let $S \subseteq R$ (that is, every attribute in s is also present in r). Then $r \div s$ is a relation on $R - S$ (that is, the result consists of attributes in $R - S$). A tuple $t \in r \div s$ iff:

- $t \in \Pi_{R-S}(r)$
- For every tuple $t_s \in s$, there is a tuple $t_r \in r$ satisfying: i) $t_r[S] = t_s[S]$ (that is, the tuples match on the attribute set S), and, ii) $t_r[R - S] = t$

Answer the following questions regarding the division operator. *Note: If your answer cannot be understood, there will be no partial marking.*

(a) **(3 points)** Design an algorithm to *efficiently* evaluate the division operator and write the pseudocode. You *should not* assume the presence of indexes.

Algorithm

~~(1) Find the set T_S of all tuples, such that $\forall t_S \in T_S$~~

(1) Iterate over all the tuples $t_S \in S$.

(2) For each t_S, iterate over all tuples $t_R \in R$ and check if $t_S[S] = t_R[S]$.

(3) If yes, take $t = \Pi_{R-S}(t_R)$, and add t to $T_{R \div S}$.

(4) $T_{R \div S}$ holds the output for the division operator.

(b) **(3 points)** Analyse the cost of your pseudocode.

Σ s_R max(|R|,|S|)

where s = number of tuples in S
R = number of tuples in R

USE FOR ROUGH WORK