CS44800 Fall 2021 Midterm 2 solutions, November 18, 2021

Prof. Chris Clifton

Turn Off Your Cell Phone. Use of any electronic device during the test is prohibited. As previously noted, you are allowed notes: Up to two sheets of 8.5x11 or A4 paper, single-sided (or one sheet double-sided).

Time will be tight. If you spend more than the recommended time on any question, **go on to the next one**. If you can't answer it in the recommended time, you are either giving too much detail or the question is material you don't know well. You can skip one or two parts and still demonstrate what I believe to be an A-level understanding of the material.

Note: It is okay to abbreviate in your answers, as long as the abbreviations are unambiguous and reasonably obvious.

In all cases, it is important that you give some idea of how you derived the answer, not simply give an answer. Setting up the derivation correctly, even if you don't carry out the calculations to get the final answer, is good for nearly full credit.

1 Index for Range Search (5 minutes, 3 points)

Consider a relation stored as a randomly ordered file for which the only index is an unclustered index on a field called sal. If you want to retrieve all records with sal > 20, is using the index likely to be better than a full table scan? Explain.

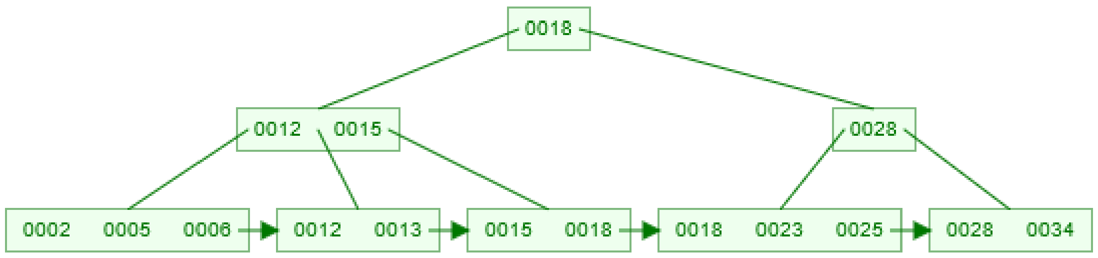
With a clustered index, we could go to salary 20 and scan through the rest of the file. But with an unclustered index (even a B+ tree), we'd have to look up each record individually. If the number of records with sal > 20 is larger than the number of blocks, then a full table scan will likely be faster, since each record retrieved could mean a block read.

Scoring: 1 for "no", 1 for showing understanding of unclustered index, 1 for showing understanding of use of index when ordered, 2 for challenge of many random lookups vs. table scan.

2 B+ Tree (15 minutes, 10 points)

Consider the following clustered index using a B+tree of order d=4 with the following assumptions:

- A left pointer in an internal node guides towards keys $<$ than its corresponding key, while a right pointer guides towards keys $\geq$.
- A leaf node underflows when the number of keys goes below $\lceil \frac{d-1}{2} \rceil$.
- An internal node underflows when the number of pointers goes below $\lceil \frac{d}{2} \rceil$.



You should assume that this is a dense index, although not shown the leaf nodes contain pointers to the block containing the actual tuples, and that the file with the tuples contains four tuples per block.

A. How many blocks would you need to access to find all records between 5 and 15 (including 5 and 15)?

This would take three blocks (root / left / left) to get to the leaf, then two more leaf reads (following the pointer chain) to get to 18 (which means you'd have past 15), for

five index blocks. But then you need to read the five records. While this would seem to be five more blocks read, since it is a clustered index and we have four tuples per block, we know that 5, 8, and 12 are in the first data block, and 13 and 15 in the second, for a total of seven blocks read.

Scoring: 1 for correct index blocks, 1 for following chain, 1 for blocks pointed to, 1 for sequential access to file blocks.

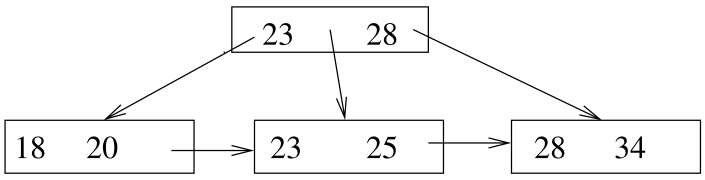
B. Delete 5 from the B+tree. Draw the resulting tree.

This simply replaces the lower left block with one containing only 0002 and 0008.

Scoring: 1 for deletion, 1 for no other changes

C. Insert 20 into the B+ tree. Draw the resulting tree.

Here the changes are all on the right side of the tree:



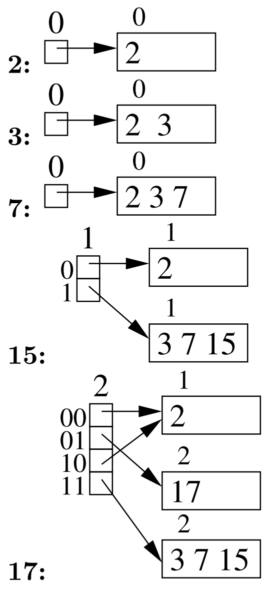
Scoring: 1 for adding in right place, 1 for correct 1st level split, 1 for correct 2nd level split, 1 for result is a valid B+ tree.

3 Extensible Hashing (10 minutes, 4 points)

Consider an extendible hashing structure where

- Initially each bucket can hold up to 3 records.
- The directory is an array of size 4.

Construct the extensible hashing structure by inserting key 2, 3, 7, 15, 17 in order, using the low order bits of the key for the hash function (e.g., the hash function for 2 would be 0, then 10, then 010, then 0010, ...; for 7 would be 1, then 11, then 111, then 0111, ...) Show each step.



Scoring: 1 for showing understanding of hash, 1 for correctly showing a split, 1 for split generating multiple pointers to same bucket, 1 for correct result.

Sample Database

The following questions are based on the schema/data below. Note that this is only a sample of the tables, the actual tables are much larger. The schema/data values shown are not strictly necessary to answer the questions, but you may find it helps you to understand the questions.

Students

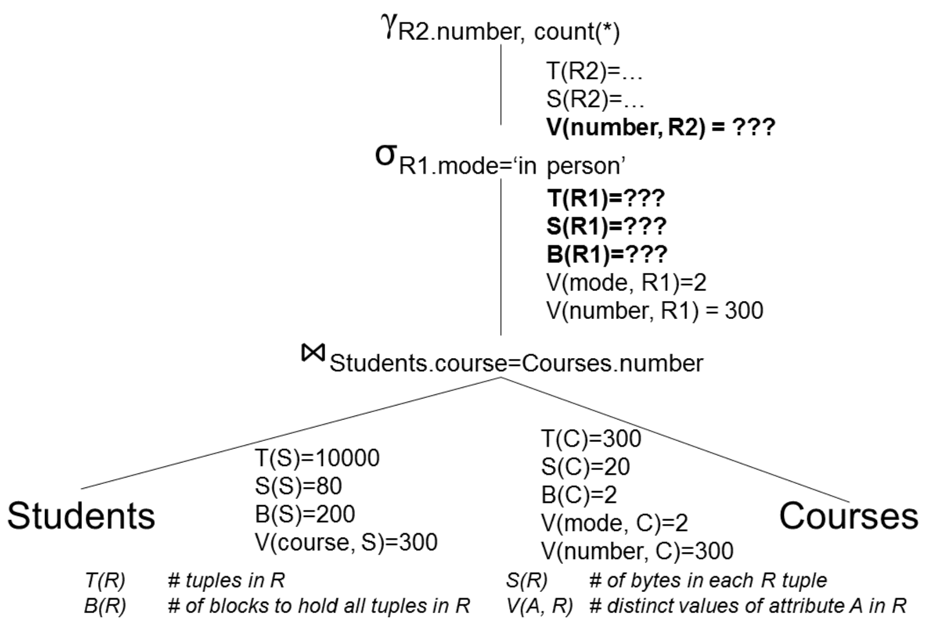
puid	*course*	*mode*
48	CS47301	remote
48	CS44800	remote
29	CS44800	campus
34	CS25000	campus
34	CS25000	campus
29	STAT35000	campus
	…	

Courses

number	*room*	*capacity*
CS25000	MATH 100	200
CS47301	LWSN 1102	30
CS44800	KRAN G016	90
CS64200	ONLINE	
STAT35000	HAAS G066	70
STAT41600	HAAS G066	70
	…	

4 Query Cost Estimation (22 minutes, 14 points)

Given the following query plan:



A. Calculate the cost estimates for the values listed as **???**. Show how you calculated these:

(a) T(R1)

$$= \frac{(T(S) \cdot T(C)}{\max(V(Course,S),V(number,C)} = \mathbf{10{,}000}$$

(b) S(R1)

$$= S(S) + S(C) = \mathbf{100}$$

(c) B(R1)
$= \frac{T(R1)*S(R1)}{blocksize}$ **= 250 (assuming 4000 byte blocks)**

(d) V(number, R2)
$= \min(T(R2), V(number, R1))$ **= 300**

Scoring: 1 for exact answer, or 0.5 for formula/discussion, 0.5 for reasonable answer.

B. Given this query plan, what join algorithm do you think would be the fastest? Assume you have indexes on Students.puid, Students.course, and Courses.number, and that you can use at most 50 buffers. Explain why you think this is the best join algorithm.

Since B(C)=2, we can do Block Nested Loop with Courses as the inner loop. This means we need one pass through Students, for 2+200 blocks read. Note that index join will require an index lookup for each tuple, which will probably be much higher - and since pretty much every tuple will have at least one match, every block will need to be read anyway - so block nested loop is almost certainly optimal.

Scoring: 2 for Block Nested Loop, 1 for hash join or merge join with explanation of need to later select for inequality, 1 for explanation showing understanding of a join algorithm, 1 for explanation based on tuple or block counts, max 3.

C. Which operations are completely blocking (you can't produce output until you have all the input), and which can be pipelined (you can produce some tuples without having all the input)? You may want to explain your answers, as some may not be a simple yes/no answer.

Pipelining: Select, if index or nested loop join in previous answer then join, if merge or hash join then group by.

Blocking: Group by unless merge or hash join, join if merge or hash join, block nested loop join with explanation showing understanding that needs to wait for full block of outer but then pipelining.

Scoring: 1 for select pipelined, 1 for correct on join, 1 for correct on group by (1 for group by blocking with merge join)

D. Give an equivalent query plan that you think should run faster. Explain why it should run faster.

Doing the select on mode before the join will reduce the size of the students table. In fact, since there are only two modes, and we see they are campus and remote, it will reduce the tuples to 0...

Scoring: 1 for query plan equivalent, 1 for pushing select, 1 for explanation

E. Do you think having an index on Students.mode would help at all for this query? Explain your answer.

In general, it would not, because there are too many tuples being returned (V(S,mode) is small). So there would be several thousand random accesses via the index, vs. 200 blocks to scan through. But in this case, since there are no modes "in person", it would immediately find there are none and speed things up.

Scoring: 1 for reason matching answer.

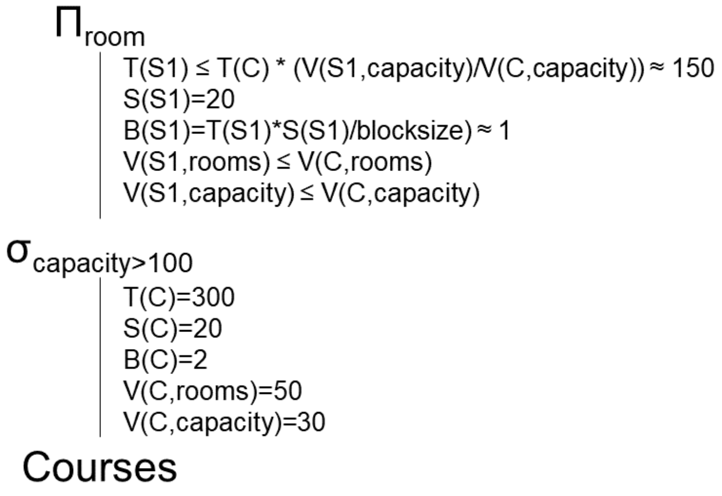
5 Query plan generation (10 minutes, 8 points)

Give a query plan, as a tree, for the SQL query:

```
SELECT room
FROM Courses
WHERE capacity > 100
```

Show cost estimates at each stage of the tree assuming Courses has 300 tuples, each tuple takes 20 bytes, blocks are 4KB, number is a key, there are 50 distinct rooms, and 30 distinct values for room capacity.

We have to make some assumptions on the selectivity of capacity¿100, which we'd typically do using max and min values. I've assumed below a min of 0 and max of 200, giving a selectivity of 0.5, but formulas shown are more generally applicable.



Scoring: 1 for select, 1 for project, 1 each for tuples, size, blocks, and all attribute values, 1 each for each of those being correct (1 for two out of three attribute selectivity)

6 Query plan selection (12 minutes, 8 points)

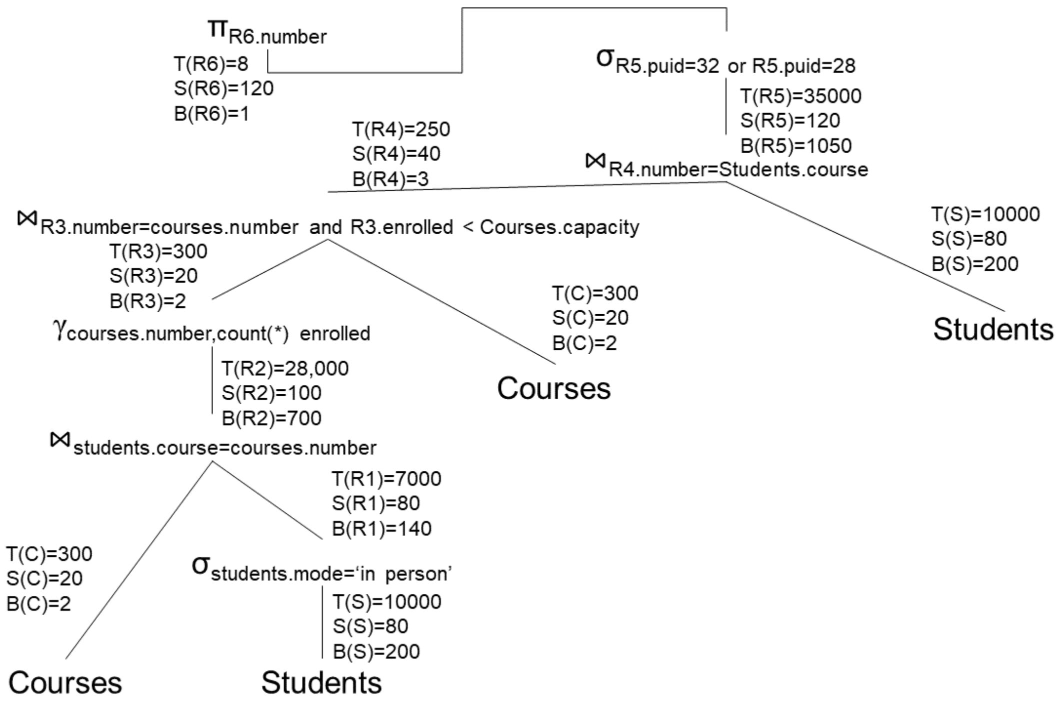
A student registering late wants to be in courses with their friends (ids 28 and 32). The following query gives courses that the friends are taking that are not full.

```
SELECT C1.number
FROM Students S1, Courses C1,
  (SELECT C2.number, count(*) enrollment
   FROM Courses C2, Students S2
   WHERE S2.mode = "campus" and S2.course=C2.number
   GROUP BY C2.number) E1
WHERE (S1.puid=28 OR S1.puid=32) AND (S1.course=C1.number) AND
      (C1.number=E1.number) AND (C1.capacity > E1.enrollment)
```

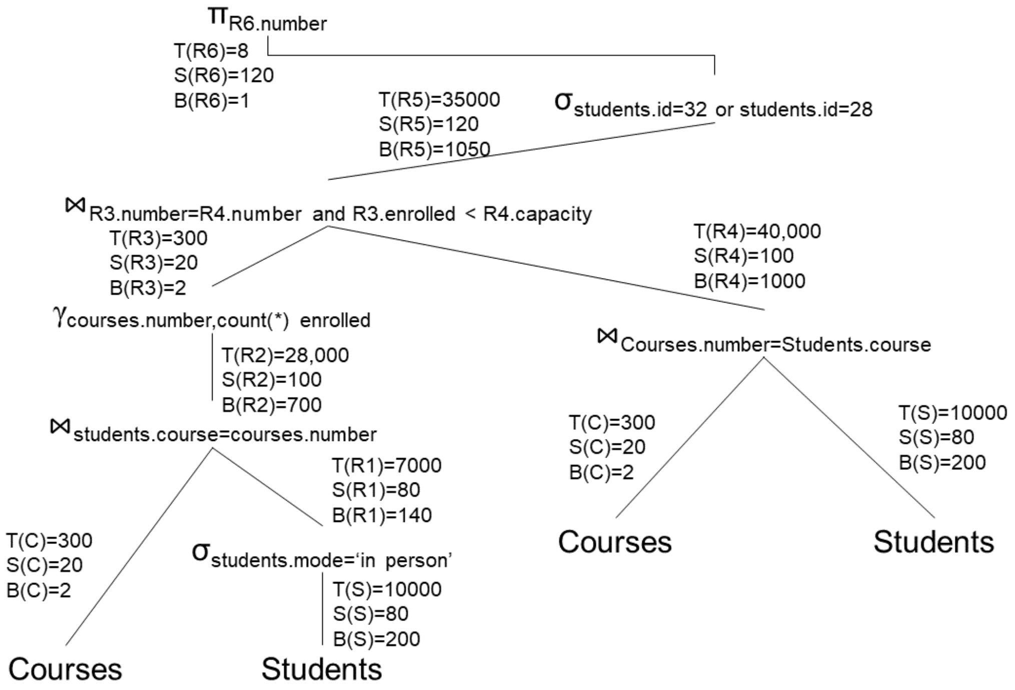
Hint: You can answer this question even if you don't understand the above query.

We give the following two query plans, with cost estimates (Don't worry about how the cost estimates were calculated, they may not be the way you would calculate them or the way you should have calculated estimates in Question 4. Just assume they are correct for this question.)

Plan A:



Plan B:



A. Which query plan do you think would be faster? Explain why.

Plan A. Both plans process roughly the same number of blocks, but R4 is 1000 blocks in plan B, but only 3 in Plan A.

Scoring: 1 for plan A, 1 for correct discussion of fewer intermediate blocks

B. Which join algorithm would you choose for the last join in Plan B (join of R3 and R4)? Assume you have indexes on Courses.number and Students.puid.

Block nested loop with R3 as the inner loop, as it fits entirely in memory. Not to mention that the join criteria involves an inequality, so to use any other algorithm we need to split into a separate select. The index on Students and Courses can't help, as we aren't joining Students and Courses, we are joining R3 and R4. And even if it did, it would probably be much slower, given the number of tuples to be looked up.

Scoring: 2 for BNL, 1 for R3 as inner, 1 for hash or merge, 1 for discussion involving costs, 1 for discussion involving why hash/merge require splitting out the inequality, why index impossible, or specific cost computation for any algorithm. Max 3.

C. Can you make a significant further improvement in any of the query plans? State what you would do and give a brief idea of how much it would improve things.

Push the select on student ID lower, as this makes the joins much smaller (only a few tuples, so everything likely to fit in memory from that point forward.)

Scoring: 1 for pushing selection or other good idea, 1 for explanation of why good, 1 for idea of cost improvement