CARNEGIE MELLON UNIVERSITY
DEPARTMENT OF COMPUTER SCIENCE
15-445/645: DATABASE SYSTEMS
PROF. ANDY PAVLO, PRACTICE XXXX

MIDTERM EXAMINATION - Solutions

IMPORTANT

- You should have 16 non-empty single-side pages, including this cover
- **No aids allowed** - only **calculator** and one 8.5 x 11 **page** (both sides)

Reminders - For your information:

- **80 minutes** duration (); Graded out of **140** points; **11** questions total
- Raise your hand, for clarifications.
- Write all your answers on the designated spaces, as **cleanly** as you can. For scratch paper, use the back pages, and the last page (and return it).

Last Name	
First Name	
AndrewId	

Question	Points	Score
Query Languages	5	
SQL	10	
File Structures	15	
Buffer Pools	10	
Linear Hashing	15	
Extendible Hashing	15	
B+Trees	15	
Sorting	15	
Join Algorithms	20	
Query Processing Models	10	
Query Execution	10	
Total:	140	

$Revision$: 2020/10/18 14:23

Question 1: Query Languages . . . . . [5 points]

Consider patients of a hospital, and their symptoms, as shown in the patient-symptom (PS) table of Figure 1.

PS

patient	symptom
Albert	sneezing
Albert	coughing
Albert	fever
Betty	sneezing
Betty	coughing
Christine	sneezing
Christine	fever
Danny	sneezing
Danny	dizziness
Ellen	sneezing
Ellen	coughing
Ellen	fever
Ellen	dizziness

Figure 1: Patients and their symptoms.

Consider the following queries in relational languages, and give their results when applied to the PS table above. Specifically:

(a) RA: $\pi_{symptom}(\sigma_{patient='Albert'}(PS))$

i. **[0 points]** OPTIONAL: What does it do, in English? (pick among the hints that follow, or modify one of them).

Solution: Lists all the symptoms of 'Albert'

ii. **[1 point]** How many columns are in the result?

ii. **1**

iii. **[2 points]** How many rows

iii. **3**

iv. **[2 points]** Give the resulting tuple(s), if any.

Solution: ('sneezing'), ('coughing'), ('fever')

Question 2: SQL . . . . . [10 points]

Grading info: -1 for one-tuple-less, for the 'how many tuples' subquestions

Given the PS table of Figure 1, show the output of each of the following SQL statements.

Hint: Try to find what each SQL statement tries to do, in English.

(a) **Selection:**

```
SELECT patient FROM PS WHERE symptom = 'fever'
```

i. **[0 points]** OPTIONAL: What does it do, in English?

Solution: Lists people that have 'fever'.

ii. **[1 point]** How many columns are in the result?

ii. **1**

iii. **[1 point]** How many rows?

iii. **3**

iv. **[1 point]** Give the resulting tuple(s), if any.

Solution: ('Albert'), ('Christine'), ('Ellen')

(b) **Join:**

```
SELECT DISTINCT P1.patient FROM PS P1, PS P2
 WHERE P1.symptom = P2.symptom
   AND P2.patient = 'Albert'
```

i. **[0 points]** OPTIONAL: What does it do, in English?

Solution: Finds distinct people, with at least one of the symptoms of Albert.

ii. **[1 point]** How many columns are in the result?

ii. **1**

iii. **[1 point]** How many rows?

iii. **5**

iv. **[1 point]** Give the resulting tuple(s), if any.

Solution: all ('Albert'), ('Betty'), ('Christine'), ('Danny'), ('Ellen')

(c) **Aggregation:**

```
SELECT patient, COUNT(*) AS n_symptoms
  FROM PS
GROUP BY patient
  HAVING n_symptoms > 2;
```

i. **[0 points]** OPTIONAL: What does it do, in English?

Solution: Lists people with more than 2 symptoms, and the count of the symptoms they have.

ii. **[1 point]** How many columns are in the result?

ii. **2**

iii. **[1 point]** How many rows?

iii. **2**

iv. **[2 points]** Give the resulting tuple(s), if any.

Solution: ('Albert', 3), ('Ellen', 4)

Question 3: File Structures ..... [15 points]

Consider a table with EMPLOYEE(ssn, name, jobcode, . . .), with

- N pages of employee records, with ssn as the primary key,
- q queries per day

As the DBA, you have to choose the appropriate file structure and (non-clustering) indices, for some settings that we list later.

Notation: For the available file structures, we use the notation FS-m-a, with m indicating the method (0/S/B/H for heap, sorted, B+tree, hash, respectively), and a for the initial of the attribute of clustering. That is, the available file structures are:

- FS-0: Heap
- FS-S-s: sorted, on ssn
- FS-B-s (FS-B-n): clustering B+tree index on ssn (name, respectively)
- FS-H-s (FS-H-n): clustering hash index on ssn (name, resp.)

For the (non-clustering = dense) indices, pick among the following choices (with similar notation: I-m-a, for method m, and attribute a):

- I-0: no non-clustering indices
- I-H-s (I-H-n) : hash index on ssn (name, resp.)
- I-B-s (I-B-n) : B+tree index on ssn (name, resp.)

Notes:

- All else being equal, prefer a hash index, over a B+tree index
- In case of a tie, report the one that needs least work (= keystrokes) on the DBA's side; if it is still a tie, report any one of the top choices, for full points.

Mark your answers, for the settings below:

Grading info: -1pt penalty, for one time only, if they mirror, eg., FS-B-s, AND I-B-s

Grading info: -1pt if B-tree, when hashing would be enough

(a) Small file, few queries:

N=10 pages, 1 query per day, exact match on ssn.

i. **[1 point]** *File structure / clustering index* - choose exactly one:
■ **FS-0** ☐ FS-S-s ☐ FS-B-s ☐ FS-B-n ☐ FS-H-s ☐ FS-H-n
Grading info: 1 of 3 pt if hash index

ii. **[2 points]** *Non-clustering index/indices* - choose one or more:
■ **I-0** ☐ I-H-s ☐ I-H-n ☐ I-B-s ☐ I-B-n

(b) Large file, many range queries:

N=1,000,000 pages, q=10,000 queries per day (range queries on ssn).

i. **[2 points]** *File structure / clustering index* - choose exactly one:
☐ FS-0 ☐ FS-S-s ■ **FS-B-s** ☐ FS-B-n ☐ FS-H-s ☐ FS-H-n

ii. **[2 points]** *Non-clustering index/indices* - choose one or more:
■ **I-0** ☐ I-H-s ☐ I-H-n ☐ I-B-s ☐ I-B-n

(c) Large file, many exact-match queries:

N=1,000,000 pages, q=10,000 queries per day (exact match on `ssn`).

i. **[2 points]** *File structure / clustering index* - choose exactly one:
☐ FS-0 ☐ FS-S-s ☐ FS-B-s ☐ FS-B-n ■ **FS-H-s** ☐ FS-H-n

ii. **[2 points]** *Non-clustering index/indices* - choose one or more:
■ **I-0** ☐ I-H-s ☐ I-H-n ☐ I-B-s ☐ I-B-n

(d) Large file, many range queries, a few exact match ones:

N=1,000,000 pages, q=100 queries per day (exact match on `ssn`); q'=20,000 queries per day (range queries on `name`).

i. **[2 points]** *File structure / clustering index* - choose exactly one:
☐ FS-0 ☐ FS-S-s ☐ FS-B-s ■ **FS-B-n** ☐ FS-H-s ☐ FS-H-n

ii. **[2 points]** *Non-clustering index/indices* - choose one or more:
☐ I-0 ■ **I-H-s** ☐ I-H-n ☐ I-B-s ☐ I-B-n
Grading info: 2 of 5 pt, if FS-H-s, I-B-n

Question 4: Buffer Pools . . . . . [10 points]

Check 'T'rue or 'F'alse for each of the following statements.

(a) **[2 points]** ■ **T** □ F Every thread in the DBMS has to pin a table heap page before it updates an existing tuple in that page. You can assume that the tuple's attributes are all fixed length and that the thread is just overwriting the existing values.

(b) **[2 points]** □ T ■ **F** If the DBMS uses a counter to track the number of threads that have pinned a page, it does not need to also use a separate boolean flag to track whether that page is dirty.

(c) **[2 points]** □ T ■ **F** The CLOCK buffer pool replacement policy always selects the least-recently used page since the last sweep.

(d) **[2 points]** ■ **T** □ F Maintaining separate buffer pools for index pages and table heap pages will prevent index pages from being swapped out memory due to sequential flooding on table scans.

(e) **[2 points]** □ T ■ **F** If a DBMS uses a *log-structured storage architecture*, it does not need to employ a buffer pool manager like in a *table heap storage architecture* to manage disk pages.

Question 5: Linear Hashing . . . . . [15 points]

GRADED BY: Jiayu Liu

Consider a hash table that operates under linear hashing. When it started, the initial hashing function was $h_0(x) = x \bmod 4$; the hash table had N_0=4 buckets (0,1,2,3), and the split pointer was s=0.

(a) After some splits, the table has N=6 buckets, numbered 0,1,. . . ,5.

i. **[2 points]** Where is the split pointer s?

i. **2**

ii. **[1 point]** How many hashing functions are active?

ii. **2**

iii. **[2 points]** Which one(s)?

Solution: $h_0(x) = x \bmod 4$, and $h_1(x) = x \bmod 8$

(b) After some other splits, the table has N=32 buckets (numbered 0,1,. . . , 31)

i. **[4 points]** Where is the split pointer s?

i. **0**

Grading info: at 0 - we have completed an expansion

ii. **[3 points]** How many hashing functions are active?

ii. **1**

iii. **[3 points]** Which one(s)?

Solution: $h(x) = x \bmod 32$

Question 6: Extendible Hashing . . . . . [15 points]

Consider the extendible hash table shown in Figure 2. This hash table has global depth g=3, with D=2^g=8 directory entries, and B=5 buckets.

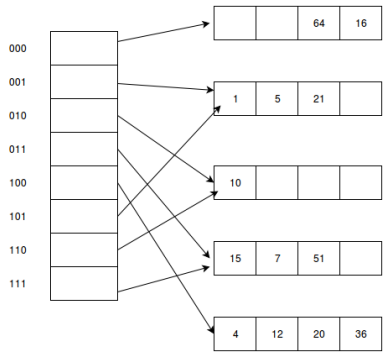


Figure 2: Extendible Hash Table

(a) **[5 points]** Imagine another hash table, with global depth g=5, and a bucket b with local depth l=2. How many directory entries point to this bucket?

☐ 0 ☐ 1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐ 7 ■ **8** ☐ 10 ☐ 12 ☐ 16 ☐ None

(b) **[5 points]** Imagine a hash table with global depth g=11. What is the *maximum* possible local depth l that a bucket could possibly have?

(b) **11**

(c) **[5 points]** Imagine a hash table with global depth g=7. What is the *minimum* possible local depth l that a bucket could possibly have?

☐ 0 ■ **1** ☐ 2 ☐ 3 ☐ 4 ☐ 7 ☐ 8

Question 7: B+Trees . . . . . [15 points]

Consider the following B+tree with of order d=2 shown in Figure 3. The maximum fanout of this tree is 5 (i.e., $2\times(d+1) = 5$).

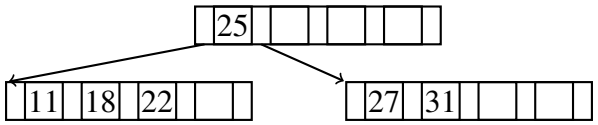


Figure 3: B+tree example

For all of the following questions, assume that we want to delete key **31** from the tree. You can assume that the DBMS will use all of the reorganization optimizations that were discussed in the lecture and textbook.

(a) **[3 points]** Before the deletion, the B+tree has 3 nodes – how many will it have after the deletion of key 31?

(a) **3 OR 1**

(b) **[4 points]** Which keys (if any) will be in the same node as key 11?

(b) **18 OR 18,22,27**

Solution: Solution #1: Key 18
Solution #2: Keys 18,22,27

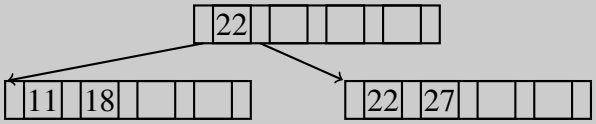
(c) **[8 points]** Draw the new tree after the deletion of key 31:

Grading info: 5/8 if correct tree with multiple guesses(?)

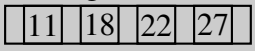
Grading info: 2/8 if explanation attempted

Grading info: 6/8 if one key is off

Solution: Solution #1: Since the deleting 31 causes the right node to be less-than half full, you can borrow a key from its sibling. This will move 22 over. You also have to update the root node to use 22 as the separator key.



Solution #2: Alternatively you can perform the merge operation on the right-most tree, causing the tree to collapse to a single node:



Question 8: Sorting . . . . . [15 points]

A DBMS is going to sort a file using external merge sort. Assume that the system uses the simplest algorithm to complete this operation. That is, it does not use double buffering nor blocked I/O, and that it uses quicksort for in-memory sorting.

We define the following variables:

- P is the number of pages in the input file.
- B is number of buffers (each able to store one page).

(a) **[5 points]** With B=75 buffers available, what is the largest file we can sort with <u>two</u> passes? That is, how many pages P will this file have?

☐ 73 ☐ 74 ☐ 75 ■ **5,550** ☐ 5,625 ☐ 7,400 ☐ 7,500

Solution: $P = B \times (B-1) = 5,500$

(b) **[5 points]** What is the smallest number of buffers B that can sort the file with P=10,000 pages in <u>two</u> passes?

☐ 10 ☐ 34 ☐ 75 ☐ 100 ■ **101** ☐ 150 ☐ 151

Solution: We want B where $P \leq B \times (B-1)$
If $B = 100$, then $10,000 > 100 \times 99 = 9,900$
If $B = 101$, then $10,000 \leq 101 \times 100 = 10,100$

(c) **[5 points]** How many pages P are in the largest file we can sort in <u>three</u> passes with B=50 buffers?

☐ 105,050 ■ **120,050** ☐ 122,500 ☐ 125,000 ☐ 150,150

Solution: $B \times (B-1)^2 = 50 \times 49^2$ = 120,050

Question 9: Join Algorithms ..... [20 points]

Consider relations $R(X, Y)$ and $S(X, Z)$ to be joined on the common attribute X. Assume that there are no indexes.

- There are $B = 30$ pages in the buffer.
- Table R spans $M = 1800$ pages with 50 tuples per page.
- Table S spans $N = 500$ pages with 100 tuples per page.

What are the I/O costs for the following joins?

- Assume the simplest cost model, where pages are read and written one at a time.
- Assume that you will need one buffer block to hold the evolving output block and one input block to hold the current input block of the inner relation.
- Ignore the cost of the final writing of the results (as all formulas in the book and the foils do).

(a) **[5 points]** Block nested loop join with R as the outer relation and S as the inner relation.
☐ 30,000 ☐ 31,800 ☐ 32,500 ☐ 33,300 ■ **34,300**

Solution: $M + \left(\left\lceil \frac{M}{B-2} \right\rceil \times N\right) = 1800 + \left(\left\lceil \frac{1800}{30-2} \right\rceil \times 500\right)$

(b) **[5 points]** Block nested loop join with S as the outer relation and R as the inner relation.
☐ 29,300 ☐ 31,100 ☐ 31,800 ■ **32,900** ☐ 34,200 ☐ 34,300

Solution: $N + \left(\left\lceil \frac{N}{B-2} \right\rceil \times M\right) = 500 + \left(\left\lceil \frac{500}{30-2} \right\rceil \times 1800\right)$

(c) **[5 points]** Sort-merge join with S as the outer relation and R as the inner relation. (*Remarks*: Ignore recursive partitioning and partially filled blocks.)
☐ 10,400 ☐ 9,500 ☐ 11,500 ☐ 14,800 ■ **15,100** ☐ 17,000

Solution:
$\left(2 \times M \times \left(1 + \left\lceil log_{B-1} \left\lceil \frac{M}{B} \right\rceil \right\rceil\right)\right) + \left(2 \times N \times \left(1 + \left\lceil log_{B-1} \left\lceil \frac{N}{B} \right\rceil \right\rceil\right)\right) + M + N =$
$(2 \times 1800 \times (1 + \lceil log_{29} 36 \rceil)) + (2 \times 500 \times (1 + \lceil log_{29} 17 \rceil)) + 1800 + 500 =$
$(2 \times 1800 \times (1 + 2)) + (2 \times 500 \times (1 + 1)) + 1800 + 500 =$
$10800 + 2000 + 1800 + 500 = 15100$

(d) **[5 points]** Hash join with S as the outer relation and R as the inner relation. (*Remarks*: Assume that the hash function splits both the outer and inner relations uniformly.)
☐ 2,300 ☐ 4,600 ■ **6,900** ☐ 7,000 ☐ 8,400

Solution: $3 \times (M + N) = 3 \times (1800 + 500)$

Question 10: Query Processing Models . . . . . [10 points]

Check 'T'rue or 'F'alse for each of the following statements.

(a) **[2 points]** □ T ■ **F** With the *iterator* processing model, each operator will immediately emit a tuple to its parent as soon its child produces a tuple.

Solution: There are pipeline breaker operators (e.g., joins, order by) that need to retrieve all of the tuples from some or all of its children operators before emitting tuples.

(b) **[2 points]** ■ **T** □ F With the *materialization* processing model, the DBMS pushes tuples up the query plan from one operator to the next.

Solution: This is the exact definition of how the materialization processing model works.

(c) **[3 points]** □ T ■ **F** Index scans are slower in the *iterator model* (compared to the materialization and vectorization models) because the DBMS has to traverse the B+Tree from the root every time the scan operator's parent calls `next()`.

Solution: The scan operator can maintain an iterator on the index that allows it to scan leaf nodes without having to traverse the tree each `next` invocation.

There is also nothing about the iterator model that says that an operator has to retrieve tuples from an index one at a time. It could be doing a leaf node scan in the access method and buffer them.

(d) **[3 points]** □ T ■ **F** Regardless of what query processing model a DBMS uses, it will always store the intermediate results of each operator in the buffer pool in case they are larger than the amount of memory available to the system.

Solution: The iterator model pipelines tuples between operators so that it does not have to store intermediate results in a buffer pool page.

Question 11: Query Execution ..... [10 points]

Consider the table EMPLOYEES(ssn,name,dept,salary), that spans P=1,000,000 pages, stored on the disk. The ssn attribute is the primary key for the table.

Assume that the DBMS is running on a machine with the following performance characteristics for its storage and memory hardware:

- 1 random disk access takes $t_r \approx 10$ milliseconds (ms)
- 1 sequential disk access takes $t_s \approx 2$ milliseconds (ms)
- 1 memory access takes $t_m < 1$ micro-second (μs)

(a) **[3 points]** Suppose you want to execute the following SQL statement:

```
SELECT * FROM EMPLOYEES WHERE ssn=33
```

Assume that

- There is a clustering, B+tree index on ssn with a height of 7.
- No other query is running at the same time.
- You just turned on the machine ("cold start"), so all buffers are empty.

How long would you expect it to take? Mark the choice that is the closest upper bound to the correct answer.

□ $\leq$1 μs □ $\leq$10 μs □ $\leq$1 ms □ $\leq$10 ms ■ $\leq$**100 ms** □ $\leq$1000 ms

Solution: You first have to travese the B+tree. Each level is a random disk read. Then you have to fetch the one page with the record you need.
$(7 \times 10\ ms) + (1 \times 10\ ms) \approx 70\ ms$
If you assume that the tuples are stored in the index leaves, then you can skip the last page read but that doesn't change the correct answer.

(b) **[3 points]** The query from the previous question (ssn=33) is executed a second time on the same system with the same indexes. Assume that

- The caches are now "warm" – all necessary pages are in main memory.
- No other query or transaction is running at the same time.

How long would you expect it to take? Again, mark the choice that is the closest upper bound to the correct answer.

□ $\leq$1 μs ■ $\leq$**10 μs** □ $\leq$1 ms □ $\leq$10 ms □ $\leq$100 ms □ $\leq$1000 ms

Solution: Even though everything is in memory, you still have to travese the B+tree. Each level is a memory access. Then you have to fetch the one page with the record you need.
$(7 \times 1\ \mu s) + (1 \times 1\ \mu s) \approx 8\ \mu s$
Again, if you assume that the tuples are stored in the index leaves, then you can skip the last page read but that doesn't change the correct answer.

(c) **[4 points]** Suppose you want to find the records of all the employees whose name includes the word "Codd", and thus you issue the query:

```
SELECT * FROM EMPLOYEES WHERE name LIKE '%Codd%'
```

Assume the following:

- There is a non-clustering B+tree index on `name`;
- You just turned on the machine ("cold start"), so all buffers are empty.
- A few more "read-only" queries are running on your machine at the same time but they are accessing different tables.

Your query takes over an hour to execute. This is way too long. Which of the following choices (if any) would make this query go faster? Select *exactly* one option from the choices below.

- ☐ Force recomputation of the DBMS's internal statistics
- ☐ Force the optimizer to use the B+tree index in the query plan
- ☐ Drop the non-clustering B+tree index on `name`, and create a *clustering* B+tree index on `name`, instead
- ☐ Drop the non-clustering B+tree index on `name`, and create a *clustering*, *hash-table* index on `name` instead
- ☐ Drop the B+tree index on `name`, and create a *radix tree* index on `name` instead
- ■ **None of the above**

Solution: The first issue is that the query is trying to do a look-up on a VARCHAR without a prefix. The B+tree index only works for exact-match and prefix queries. Thus, it is not able to use the existing B+tree index for this query and thus it has to perform a sequential scan across the entire table.

Here are the reasons the given options will not help:

1. Recomputing the DBMS's internal statistics will not cause the sequential scan to magically go faster.
2. Forcing the DBMS's optimizer to choose to perform a index scan on the B+tree is the worst option here because it is a non-clustered index. This means that it could possible perform a page fetch for every tuple it checks since the tuples are not stored in the same order on pages that they appear in the index's leaf nodes.
3. Switching to a clustered B+tree index will make an index scan go just as fast a sequential scan on the table heap, so this option doesn't help.
4. Switching to a clustered hash table index does not help at all either, since a hash tables only work for exact matches on keys.
5. The radix tree is a more nuanced answer. Since the query does not have the prefix before the string "Codd", the DBMS has to evaluate every possible terminal node in the radix tree. This is because the DBMS cannot tell whether the terminal node will contain the word "Codd" without examining it. This is faster than a sequential scan, but in the worst case scenario, all of the names attributes contain "Codd",

which means that the DBMS will have to retrieve every tuple. And since it is not a clustering index, it will have the same problem of using the index scan on the non-clustered B+tree index.

Grading info: 2 *points if they attempt to clarify why or why not the radix tree may not help.*