CS 186
Fall 2022

Introduction to Database Systems

Midterm

INSTRUCTIONS

This is your exam. Complete it either at exam.cs61a.org or, if that doesn't work, by emailing course staff with your solutions before the exam deadline.

This exam is intended for the student with email address `<EMAILADDRESS>`. If this is not your email address, notify course staff immediately, as each exam is different. Do not distribute this exam PDF even after the exam ends, as some students may be taking the exam in a different time zone.

For questions with **circular bubbles**, you should select exactly *one* choice.

○ You must choose either this option

○ Or this one, but not both!

For questions with **square checkboxes**, you may select *multiple* choices.

☐ You could select this choice.

☐ You could select this one too!

You may start your exam now. Your exam is due at <DEADLINE> Pacific Time. Go to the next page to begin.

Preliminaries

1 CS 186 Fall 2022 Midterm 1

`You have 110 minutes to complete the midterm. Do not share this exam until solutions are released.`

1.0.1 Contents:

- The midterm has 6 questions, each with multiple parts, and worth a total of 100 points.

1.0.2 Aids:

- You may use 1 page (double sided) of handwritten notes as well as a calculator.
- You must work individually on this exam.

1.0.3 Grading Notes:

- All I/Os must be written as integers. There is no such thing as 1.02 I/Os – that is actually 2 I/Os.
- 1 KB = 1024 bytes. We will be using powers of 2, not powers of 10.
- Unsimplified answers, like those left in log format, will receive a point penalty.

(a) **(0.5 pt)** What is your full name?

(b) What is your student ID number?

(c) SID of the person to your left (Put None if you are taking the exam remotely):

(d) SID of the person to your right (Put None if you are taking the exam remotely):

1. (15 points) Yes, I do a little bit of investing. . .

It is October 2022 and you see news about the economy heading towards a recession. Curious and still optimistic, you see this as an opportunity to buy the dip (buy stocks when the price is low)! However, before buying, it is crucial that you do your research on what stocks are promising and can recover.

(a) (3 points) Reputable Analysts

You are given the following tables:

```
/* This table is a list of stocks and their stock symbol and the
company they correspond to. */
CREATE TABLE Stocks (
    stock_id INTEGER PRIMARY KEY,
    company_id INTEGER REFERENCES Companies,
    stock_symbol VARCHAR(4) NOT NULL
);

/* This table contains info about companies: the company name,
whether or not the company is hiring and what type of business the company does. */
CREATE TABLE Companies (
    company_id INTEGER PRIMARY KEY,
    company_name VARCHAR(20) NOT NULL,
    is_hiring BOOLEAN NOT NULL,
    business_type VARCHAR(20) NOT NULL
);

/* This table contains info about analysts: the company they work for,
their first name, last name and whether or not they are reputable. */
CREATE TABLE Analysts (
    analyst_id INTEGER PRIMARY KEY,
    company_id INTEGER REFERENCES Companies,
    first_name VARCHAR(20) NOT NULL,
    last_name VARCHAR(20) NOT NULL,
    is_reputable BOOLEAN NOT NULL
);

/* This table contains info about the analysis an analyst does, which includes the rating an
analyst gives a stock and the date the analysis was given (year, month, day). */
CREATE TABLE Analysis (
    analysis_id INTEGER PRIMARY KEY,
    analyst_id INTEGER NOT NULL REFERENCES Analyst,
    stock_id INTEGER NOT NULL REFERENCES Stocks,
    rating INTEGER NOT NULL,
    year INTEGER NOT NULL,
    month INTEGER NOT NULL,
    day INTEGER NOT NULL
);
```

Assume there is a 1:1 correlation between `stock_id` and `company_id` (meaning every entry in the Stocks table corresponds to a unique entry in the Companies table).

i. (1 pt) You want to find the name of all the companies with at least one reputable analyst. If there are multiple companies with the same name with at least one reputable analyst, we want the same company name to be returned multiple times.

Does the following query achieve this?

```
SELECT DISTINCT Companies.company_name
FROM Analysis, Stocks, Companies
WHERE Analysis.stock_id = Stocks.stock_id
AND Stocks.company_id = Companies.company_id
AND Analysis.rating <= 2;
```

◯ Yes

[✓ No]

We did not include the Analyst table in our query. We need to use the Analysts table because we need to use the is_reputable field.

ii. (1 pt) Same question but instead for this query:

```
SELECT DISTINCT Companies.company_name
FROM Companies INNER JOIN
(SELECT Analysts.company_id
   FROM Analysts
   WHERE Analysts.is_reputable
   GROUP BY Analysts.company_id) AS c
ON Companies.company_id = c.company_id;
```

◯ Yes

[✓ No]

If two companies have the same name and both of these companies have reputable analysts, this query will only output the company name once. It should be output twice.

iii. (1 pt) Same question but instead for this query:

```
SELECT Companies.company_name
FROM (SELECT company_id
   FROM Companies

   EXCEPT

   SELECT DISTINCT company_id
   FROM Analysts
   WHERE NOT is_reputable) AS c
INNER JOIN Companies
ON c.company_id = Companies.company_id;
```

◯ Yes

[✓ No]

In the case where a company has two analysts, one which is reputable and another which is not, the company_id will then be excluded from the inner query (SELECT company_id FROM Companies) EXCEPT (SELECT DISTINCT company_id FROM Analyst WHERE NOT is_reputable).

(b) (3 points) Yes I dabble in crypto. . .

i. Your friend tells you about a good analyst with the first name “Raymond”. However, there are many analysts named “Raymond”! You now want to find the **number of companies** that had a **reputable** analyst with the first name **“Raymond”** who performed some analysis for said company in the year **2022**. If the analyst did not analyze any companies in 2022, your query should output 0 for that analyst. Queries should return the `analyst_id` and the number of companies the analyst covered.

Which of the following queries achieve this? **There can be zero, one, or more than one correct queries. In the case that there are zero correct queries, mark D. None of the above.**

Assume there is a 1:1 correlation between `stock_id` and `company_id` (meaning every entry in the Stocks table corresponds to a unique entry in the Companies table).

```
A. WITH Raymond AS (
      SELECT Analysts.analyst_id AS analyst_id
      FROM Analysts
      WHERE Analysts.is_reputable AND Analysts.first_name = "Raymond"),

   Raymond_stocks AS (
      SELECT DISTINCT Raymond.analyst_id AS analyst_id, Analysis.stock_id AS stock_id
      FROM Raymond LEFT OUTER JOIN Analysis
      ON Analysis.year = 2022 AND Raymond.analyst_id = Analysis.analyst_id)

   SELECT Raymond_stocks.analyst_id, COUNT(Raymond_stocks.stock_id)
   FROM Raymond_stocks
   GROUP BY Raymond_stocks.analyst_id;

B. SELECT Raymond.analyst_id, COUNT(Analysis.stock_id)
   FROM Analysis INNER JOIN
   (SELECT analyst_id FROM Analysts
       WHERE first_name = "Raymond" AND is_reputable)
   AS Raymond
   ON Analysis.analyst_id = Raymond.analyst_id AND Analysis.year = 2022
   GROUP BY Raymond.analyst_id;

C. WITH num_analyzed AS (
      SELECT Stocks.company_id as company_id, COUNT(Analysis.analyst_id) as count
      FROM Stocks LEFT OUTER JOIN Analysis
      ON Stocks.stock_id = Analysis.stock_id AND Analysis.year = 2022
      GROUP BY Stocks.company_id
      HAVING COUNT(Analysis.analyst_id) >= 1)

   SELECT Analysts.analyst_id, num_analyzed.count
   FROM Analysts INNER JOIN num_analyzed
   ON Analysts.company_id = num_analyzed.company_id
   AND Analysts.first_name = "Raymond" AND Analysts.is_reputable;

D. None of the above.
```

■ A

☐ B

☐ C

☐ D

Query A: The Raymond table first gets the analyst_id of all analysts that are reputable and are called “Raymond Ji”. The Raymond_stocks table gets all unique stock_ids that an analyst from the Raymond table analyzed in 2022. The DISTINCT keyword is important here because Raymond could have had multiple analyses of the same stock in 2022. LEFT OUTER JOIN is important here because it is possible an analyst did not have any analysis in 2022. We then GROUP BY and count the number of stocks an analyst analyzed. This is guaranteed to be the total number of companies because `stock_id` and `company_id` have a 1:1 correlation.

Query B: Using an INNER JOIN is incorrect because our output will then not have analysts who analyzed 0 companies.

Query C: The num_analyzed table finds the number of times a company is analyzed in 2022 by analysts. Our output is returning an analyst and the number of times the company they work for is analyzed, which is not what we are asking for.

(c) (9 points) Performance Review

To help you better understand an analyst's performance and whether or not they are reputable, you want to calculate the variance of an analyst's ratings for each specific stock. An analyst can analyze a stock multiple times, and each stock can be analyzed by multiple analysts.

For example, we want our query's output to look like this:

analyst_id	stock_id	variance
analyst_id1	stock_id1	x
analyst_id1	stock_id2	y
analyst_id2	stock_id1	z

The formula for variance is: $\frac{\sum(X-\mu)^2}{N}$

where X = rating for a stock by an analyst, μ = average rating for a stock by an analyst and N = number of times a stock was rated by an analyst.

```
WITH
stock_mu as (
   SELECT Analysis.analyst_id as analyst_id, Analysis.stock_id as stock_id,
   _____(1)_____ as average
   FROM Analysis
   GROUP BY _____(2)_____
),

differences as (
   SELECT Analysis.analyst_id as analyst_id, Analysis.stock_id as stock_id,
   (Analysis.rating - _____(3)_____) * (Analysis.rating - _____(3)_____) as diff_squared
   FROM Analysis, _____(4)_____
   WHERE _____(5)_____
)

SELECT differences.analyst_id as analyst_id, differences.stock_id as stock_id,
   _____(6)_____ / _____(7)_____ as variance
FROM differences
GROUP BY _____(2)_____;
```

i. **(1 pt)** What belongs in all the blanks labeled (1)?

AVG(Analysis.rating)

ii. **(1 pt)** What belongs in all the blanks labeled (2)?

analyst_id, stock_id

iii. **(1 pt)** What belongs in all the blanks labeled (3)?

stock_mu.average

iv. (1 pt) What belongs in all the blanks labeled (4)?

stock_mu

v. (1 pt) What belongs in all the blanks labeled (5)?

Analysis.analyst_id = stock_mu.analyst_id AND Analysis.stock_id = stock_mu.stock_id

vi. (2 pt) What belongs in all the blanks labeled (6)?

SUM(differences.diff_squared)

vii. (2 pt) What belongs in all the blanks labeled (7)?

COUNT(differences.diff_squared)

2. (19.5 points) Hashing

(a) (15 points) Inventory

Suppose we have the following table:

```
CREATE TABLE Inventory (
   productid INTEGER PRIMARY KEY,
   category VARCHAR NOT NULL,
   quantity INT NOT NULL
);
```

This table contains **100,000** records split across **700** pages. For all parts, assume we have **10** buffer pages. Also assume that the conquer (build) phase of hashing is always performed.

For the first 10 parts (part i - part x) **only**, assume we are partitioning the table on `productid` and that the first partitioning pass uses an uneven hash function that places **210** pages in the first partition, **105** pages in each of the second and third partitions, and evenly spreads the remaining pages among the other partitions. All following passes after the first make use of independent, perfectly uniform hash functions.

i. (0.5 pt) How many pages will be read in the first partitioning pass?

700

The table contains 700 pages, all of which will be read in the first partitioning pass.

ii. (0.5 pt) How many pages will be written in the first partitioning pass?

702

The first three partitions have 210, 105, and 105 pages written out, respectively. Per the question, the remaining 280 pages will be split among the remaining 6 partitions, each of which will have ceil(280 / 6) = 47 pages.

210 + (105*2 + 47*6) = 702 IOs

iii. (1 pt) Consider the first partition generated by the first pass. How many additional (not including the first) partitioning passes will we need to fully hash this partition?

2

The first partition has 210 pages. The second partitioning pass will create partitions of size ceil(210/9) = 24 pages. This is still larger than B=10, so an additional partitioning pass will have to be performed on the first partition.

Partial credit (0.25 points) was given for the answer of 3 which included the conquer pass.

iv. (1 pt) Now consider the second partition generated by the first pass. How many additional (not including the first) partitioning passes will we need to fully hash this partition?

2

The second partition has 105 pages. The second partitioning pass will create partitions of size ceil(105/9) = 12 pages. This is still larger than B=10, so an additional partitioning pass will have to be performed on the second partition.

Partial credit (0.25 points) was given for the answer of 3 which included the conquer pass.

v. (1 pt) Now consider the **fourth** partition generated by the first pass. How many additional (not including the first) partitioning passes will we need to fully hash this partition?

1

The fourth partition has 47 pages. The second partitioning pass will create partitions of size ceil(47/9) = 6 pages. This is less than or equal to B=10, so no further partitioning passes will be required.

Partial credit (0.25 points) was given for the answer of 2 which included the conquer pass.

vi. (2 pt) In total, across all applicable partitions, how many additional IOs will the second partitioning pass take? Don't include IOs from writing the output of the first pass to disk.

1458

The output of the first partitioning pass must first be read in, for a total of 702 IOs. The first partition will be split into 9 partitions of ceil(210 / 9) = 24 pages. The second and third partitions will each be split into 9 partitions of ceil(105 / 9) = 12 pages. The remaining six partitions will each be split into 9 partitions of ceil(47 / 9) = 6 pages. All these new partitions must be written out.

702 + (24*9 + 2*12*9 + 6*6*9) = 1458 IOs

vii. (2 pt) In total, across all partitions, how many additional IOs will any remaining partitioning passes take (if there are any left)? Don't include IOs from writing the output of the second pass to disk.

If you are not confident in your answers to previous parts, for **quarter credit**, assume the second partitioning pass returned 27 partitions in which the first three partitions each have 21 pages, the next three each have 15 pages, the next two each have 10 pages, and the rest each have 6 pages.

999

Any partition of more than B=10 pages from the output of the second partitioning pass (and only these partitions) must be recursively partitioned again. This encompasses 9 partitions of 24 pages each and 18 partitions of 12 pages each. First, read in all these partitions, for a total of 24 * 9 + 2 * 12 * 9 = 432 IOs. The 24 page partitions (of which there are 9) will each be split into 9 partitions of ceil(24 / 9) = 3 pages. The 12 page partitions (of which there are 18) will each be split into 9 partitions of ceil(12 / 9) = 2 pages. All these new partitions must be written out.

(24*9 + 2*12*9) + (3*9*9 + 2*2*9*9) = 999 IOs

The quarter credit option has an answer of 243 IOs.

viii. (2 pt) How many IOs will the conquer pass take?

If you are not confident in your answers to previous parts, for **quarter credit**, assume the final result of the partitioning stage is 115 partitions with the first ten partitions each having 8 pages, the next seven each having 7 pages, the next six each having 5 pages, and the rest each having 3 pages.

1782

Every partition resulting from the partitioning passes has to be read in and written out. From the second partitioning pass there were 6*9 partitions of 6 pages each. From the third partitioning pass there were 9*9 partitions of 3 pages each and 2*9*9 partitions of 2 pages each.

2 * (6*6*9 + 3*9*9 + 2*2*9*9) = 1782 IOs

The quarter credit option has an answer of 870 IOs.

ix. (1 pt) Adding everything up, how many IOs will this overall hashing process take from start to finish?

5641

x. (1 pt) In order to successfully perform this entire hashing process from the previous parts, how many additional hash functions will we need, **not** including the one used for the first partitioning pass?

3

Every partitioning pass and the conquer pass each need their own hash function. Thus, not including the first partitioning pass, there are 2 more partitioning passes and 1 conquer pass. This means a total of 3 additional hash functions are needed.

xi. (1 pt) Assuming we now have access to independent, perfectly uniform hash functions for every pass, including the first. How many IOs will it take to hash this table on `productid`?

4291

In the first partitioning pass, the entire table (700 pages) is read in. It is split into 9 partitions of size ceil(700 / 9) = 78 pages. These partitions must be written out to complete the first partitioning pass, which totals 700 + 78*9 = 1402 IOs.

In the second partitioning pass, the previous partitions (each of which has 78 pages) are all read in. They are each split into 9 partitions of ceil(78 / 9) = 9 pages. These partitions must then be written out to complete the second partitioning pass, which totals 78*9 + 9*9*9 = 1431 IOs.

All partitions are now less than or equal to B=10 pages, so the conquer phase can begin. Read in every partition that resulted from the partitioning stage, construct the in-memory hash table, and then write it back out. This totals 2 * (9*9*9) = 1458 IOs.

(700 + 78*9) + (78*9 + 9*9*9) + 2 * (9*9*9) = 4291 IOs

xii. (1 pt) Assuming we again have access to independent, perfectly uniform hash functions for every pass, what is the maximum number of pages we can hash using at most 4 distinct hash functions if we are hashing on `productid`?

7290

The conquer pass and every partitioning pass each require their own hash function. Thus, given 4 distinct hash functions, 3 partitioning passes and 1 conquer pass can be performed. Each partitioning pass splits each of the input partitions into 9 partitions, and the conquer pass can operate over partitions of up to 10 pages in size.

9 * 9 * 9 * 10 = 7290 pages

xiii. (1 pt) Assuming we again have access to independent, perfectly uniform hash functions for every pass, what is the maximum number of pages we can hash using at most 4 distinct hash functions if we are now hashing on `category`? Assume that `category` takes on one of 5 unique values and has a uniform distribution.

50

The conquer pass can operate on partitions of sizes up to 10 pages each. Thus, since keys with the same value are never split into different partitions by hashing, in the best case, each of the 5 unique values will be placed into its own partition. As long as each of these partitions has up to 10 pages, hashing can be performed as is.

Thus, up to 5 * 10 pages of data can be hashed using 1 partitioning pass and 1 conquer pass. Under this best-case scenario, the first partitioning pass will place each unique value into its own partition, all of which will be exactly 10 pages (and thus can be processed by a conquer pass with B=10).

(b) (4.5 points) More hashing questions

i. (2 pt) Consider the external hashing algorithm presented in lecture. Will the algorithm terminate if we use the same hash function for all partitioning passes? If so, write "yes" below. Otherwise, write "no" and explain in one sentence (less than 25 words) why not.

No, every record would get put back into the same bucket as in the previous pass.

ii. (2.5 pt) For which of the following tasks can hashing be used to implement? **There can be zero, one, or more than one correct choice. In the case that there are zero correct choices, mark F. None of the above.**

☐ A. Ordering a table by a key

■ B. Grouping a table by a key

■ C. Removing duplicates from a table

☐ D. Filtering a table on an arbitrary condition

■ E. Creating a lookup table to quickly find records with a key

☐ F. None of the above

A: False. Hashing arbitrarily orders data onto partitions.

B: True. Hashing groups data with the same value, so it can be used for GROUP BY's.

C: True. Hashing groups data with the same value, so it can be used to quickly find (and remove) duplicate values.

D: False. Hashing cannot filter data on arbitrary predicates (such as inequalities).

E: True. Once the data is hashed, individual values can be quickly looked up by pulling the relevant partition into memory.

3. (13 points) Disk and Files

Joy is writing letters to all her friends and professors as she prepares to graduate. She keeps track of the letters she wrote in a database.

This is what the schema of the table looks like.

```
CREATE TABLE Letters (
   letter_id INTEGER PRIMARY KEY,
   recipient_name VARCHAR(30),
   word_count INTEGER NOT NULL,
   description VARCHAR(40),
   letter_body VARCHAR(60)
);
```

For all parts of this question dealing with this table, assume the following information:

- Record headers contain a bitmap. Bitmaps should be as small as possible, rounded up to the nearest byte.
- Each data page is 500 B.
- The Letters table is stored as a heap file using the page directory implementation.
- Each header page has 7 entries. There are 120 data pages.
- If a question requires the use of slotted pages, the implementation of slotted pages will be the same as seen in lecture, discussion, and course notes.

(a) **(3 pt)** What is the maximum number of records that can fit onto one data page?

16

i. Ideal record size would be when the variable length fields take up 0 bytes. Therefore, the size of a record will be 21 bytes (1 byte for the bitmap, 12 bytes for the variable length field's pointer, 4 bytes for letter_id, 4 bytes for word_count). We store variable length records on slotted pages. In discussion, slotted pages have a free space pointer, a slot count, and [record length, record pointer] per record. Therefore, the number of records that can be stored on one page is as follows: Floor((500 - 8) / (21 + 8)) = 16 records.

(b) **(5 points)**

For the following questions, assume the queries are run independent of each other and that the records are stored as variable length records. Also, the buffer is large enough to hold all data and header pages, and starts empty for each question.

i. (3 pt) What is the minimum I/O cost to execute the following query successfully? Assume a successful insert.

INSERT INTO Letters VALUES (186, "Devang", 6, " ", "Hello!");

Hint: Insertion of a new record requires checking uniqueness of its primary key against all existing records in the database.

140

Inserting a new record will require checking if the letter_id (primary key) is unique. As a result, all data pages must be checked to ensure there is no duplicate. There are ceil(120/7) = 18 header pages and there are 120 data pages in total. The I/O cost will be 18 I/Os to read in the header pages, 120 I/Os to read in the data pages, 1 I/O to write to a data page with free space, and 1 I/O to update the page header with the relevant metadata. Therefore, the total cost will be 140 I/Os.

Partial credit (1 point) was given for an answer of 4, which does not check for primary key uniqueness.

ii. (2 pt) Assume records that have a letter_id in the range of [1, 20] fall on 20 different data pages, where each letter_id falls on a unique page. What is the minimum I/O cost for the following query?

SELECT recipient_name WHERE letter_id > 3 and letter_id <= 18;

18

Since letter_id is a primary key, we know it is unique for every entry, and there are exactly 15 records that fall in this range. From the problem statement, we know all 15 records fall on different data pages. In the best case, the 15 data pages are stored on the first 3 header pages. Therefore, the total I/O cost will be 18 (3 I/Os to read in the header pages, and 15 I/Os to read in the data pages).

(c) (3 points)

For each of the following questions, select the page and file format combination out of these four choices that is the best for her use case, considering space and I/O costs. Assume each question is independent of ones before it, i.e., we do not need to select the page and file format combination which optimizes for both our current operation and those before it.

A. packed, heap file

B. unpacked, heap file

C. packed, sorted file on recipient_name

D. unpacked, sorted file on recipient_name

i. (1 pt) Joy realizes she also wants to write letters to all her students and needs to insert hundreds of new records into the database. Assume that letters are retrieved by name.

[✓ A]

○ B

○ C

○ D

Frequent inserts, no deletion so packed file would save the most space

ii. (1 pt) After Joy finishes writing all letters, she wants to read and modify the contents of letters written to her friends. She needs to be able to retrieve the corresponding letters from the table by the person's name and modify them. Recall this question is independent of the previous one, i.e. she no longer inserts records into the database.

○ A

○ B

○ C

[✓ D]

Sorted file can help find the letter she wishes to update. Unpacked allows making updates and deletions without having to shift other records to maintain a packed layout.

iii. (1 pt) Before sending out the letters, she wants to scan through all records in alphabetical order of the recipient_name to remind herself who she has written letters to. Recall this question is independent of the previous ones, i.e. she no longer inserts or modifies records in the database.

○ A

○ B

[✓ C]

○ D

Need a sorted file on recipient_name to scan efficiently in alphabetical order. No deletion or updates, so packed layout would save the most space

(d) **(2 pt)** Which of the following statements are true? Assume the average case for all choices. **There can be zero, one, or more than one correct choices. In the case that there are zero correct choices, mark E. None of the above.**

- [✓ A. Insertion and deletion are faster for unpacked pages than packed pages.]
- ☐ B. I/O cost of performing a full scan on a sorted file is higher than performing a full scan on a heap file.
- [✓ C. If the page directory fits in memory, the page directory implementation of a heap file incurs less cost to find a page with enough free space than the linked list implementation.]
- ☐ D. Magnetic disks are organized into cells and support fast random reads.
- ☐ E. None of the above statements are true.

A: The original answer is True, but False was also given credit for this question as insertions are not necessarily faster for unpacked pages in the general case.

B: False, I/O costs are the same for full scan.

C: True.

D: False, SSDs/flash are organized into cells and support fast random reads.

E: False.

16. (16 points) Sorting

Ross, Joey, and Phoebe are friends running sorting tasks on the same machine. Ross's table has 1500 pages, Joey's table has 1000 pages, and Phoebe's table has 500 pages.

- Their tables are sorted simultaneously (i.e. any pass starts at the same time for each table).
- At the start of every pass after pass 0, the machine allocates half of the available buffer pages to the table with the most number of sorted runs, one third of the available buffer pages to the table with the second most number of sorted runs, and the rest (one sixth) to the table with the least number of sorted runs.
- The machine has 30 buffer pages, split evenly among Ross, Joey and Phoebe for pass 0.

(a) (1 pt) How many sorted runs does Ross have after pass 0?

150

(b) (1 pt) How many sorted runs does Ross have after pass 1?

11

(c) (1 pt) How many sorted runs does Joey have after pass 1?

12

(d) (1 pt) How many sorted runs does Phoebe have after pass 1?

13

(e) (1 pt) How many sorted runs does Ross have after pass 2?

3

(f) (1 pt) How many sorted runs does Joey have after pass 2?

2

(g) (1 pt) How many sorted runs does Phoebe have after pass 2?

1

(h) (2 pt) How many I/Os does Ross's sort take?

12000

Note: partial credit (0.5 points) was given for an answer of 6000.

(i) (2 pt) How many I/Os does Joey's sort take?

8000

Note: partial credit (0.5 points) was given for an answer of 8000.

(j) (2 pt) How many I/Os does Phoebe's sort take?

3000

Note: partial credit (0.5 points) was given for an answer of 1500.

(k) (3 pt) Now suppose Ross, Joey, and Phoebe combine all their tables and sort using all of the machine's available buffer pages. How many I/Os does this sort take? (This question is independent of the previous subparts. Assume all tables are unsorted.)

18000

Note: partial credit (0.5 points) was given for an answer of 9000.

17. (13 points) Bluffer Management

(a) **(3 pt)** Which of the following statements are true? **There can be zero, one, or more than one correct choices. In the case that there are zero correct choices, mark G. None of the above.**

☐ A. In the MRU eviction policy, the page that gets evicted is always the one most recently loaded into memory.

☐ B. After a requester removes its pin from the page, the page is always free to be removed by the buffer manager.

☐ C. One benefit of the clock algorithm is that it avoids sequential flooding.

☐ D. The reference bit in the clock policy can be utilized in place of the pin count that the other replacement policies require.

☐ E. For any sequential access pattern, MRU will always perform better than LRU in terms of cache hit rate.

☐ F. If the size of the buffer is the same as the size of the disk, we may still be susceptible to sequential flooding.

[✓ G. None of the above.]

False. The page that gets replaced is actually the page most recently *used*, which may differ from the one most recently *loaded* in to the buffer pool. False. Another requester may have also pinned the page so it isn't free to be evicted. False. The clock algorithm is still susceptible to sequential flooding. It is used in favor of LRU because its simpler to implement. False. They serve totally different purposes. Reference bit helps approximate LRU, pin count determines if a page is eligible for eviction. False. LRU and MRU can have the same hit rate if the length of the access pattern is less than the buffer size or the buffer can only fit one page. False. Since the size of the buffer and disk are the same, all pages from disk fit into the buffer - as such we will not need to evict any pages from memory.

From the above answers, it seems the writer of this question sure loves to bluff.

(b) **(5 points)**

For the next group of questions, assume you have 3 (initially empty) buffer frames and all accesses are unpinned immediately.

i. **(1.5 pt)** Consider the following page access pattern:

A , B , C , D , E , C , C , E , D

Which policy(s) has the least amount of cache hits?

☐ A. LRU

[✓ B. MRU]

☐ C. Clock

ii. **(1 pt)** How many hits does the cache receive using the worst performing eviction policy(s)?

MRU: after the first two accesses to A and B, the page in the last buffer frame is repeatedly evicted, aside from a singular hit to page C.

iii. (1.5 pt) Now consider the following page access pattern:

A , B , C , D , D , C , B , A

Which policy(s) has the highest amount of cache hits?

- [✓ A. LRU]
- [✓ B. MRU]
- [✓ C. Clock]

iv. (1 pt) How many hits does the cache receive using the best performing eviction policy(s)?

LRU, MRU, and Clock perform equally - all achieve 3 cache hits.

(c) (5 points)

Consider a new eviction policy: the LFU (Least Frequently Used) policy. Whenever a page is loaded into the buffer, a value representing the page's hit counter is set to 1. Every time the page is requested afterwards, the counter is incremented. The policy chooses to evict the page with the lowest hit counter. Ties are broken using LRU. Still assume you have 3 (initially empty) buffer frames and all accesses are unpinned immediately.

Vik tells the rest of course staff that this new LFU policy outerperforms LRU. Is he bluffing (bluffing means not telling the truth) or not?

i. (1 pt) Consider the following page access pattern:

A , B , C , B , D , A , C , D , A

How many hits does the cache receive using LRU?

3

ii. (2 pt) How many hits does the cache receive using LFU?

1

iii. (2 pt) Identify the problem that the policy with the lower number of cache hits suffers with that the other policy avoids in the access pattern above? Answer using at most a sentence.

LFU suffers from sequential flooding.

After the buffer frame for B's counter increases to 2 (after the 4th access), the other frames are left with a counter of 1. They repeatedly exchange turns being "least recently used" to break the tie for eviction and both frames continue to stay at a counter of 1. This access pattern effectively turns into sequential flooding, cycling through 3 data pages with an effective buffer size of 2 since B's buffer frame is never touched.

Edit: Any mention of "difficulty evicting frame B" or similar also receives credit. However, simply mentioning "popularity shift" without mentioning how that specifically affects the cache will not. Popularity shift is a property of the access pattern, not a specific problem that one policy faces that the other avoids.

18. (23 points) B+ Trees

Justin recently got fascinated by the HBO series “House of the Dragon”, especially the history of the people in the movie. As such, he created a table of characters in the series.

```
CREATE TABLE characters {
    tid INTEGER PRIMARY KEY,
    firstname TEXT,
    lastname TEXT,
    born_year INTEGER,
    death_year INTEGER,
    house TEXT,
    culture TEXT,
    ... // other fields
}
```

For the entire B+ Tree question, assume we have unlimited buffer pages

(a) (3 points) Targaryen Dynasty

To make retrieving a character's record faster, he created an index A on the table.

Index A:

- Alt 1 Tree built on tid with height = 3, order = 3

i. (1 pt) Is the following statement True or False: tid can have a null value.

○ True

[✓ False]

tid is a primary key, and the primary key cannot be a null value.

ii. (2 pt) We want to retrieve a record with tid = 1000, what's the minimum number of I/Os to achieve this?

4 I/Os

Traversing the inner nodes takes 3 I/Os to read, and since it's alt 1, the leaf node contains the records, so it takes in total 4 I/Os to retrieve the record.

(b) (20 points) RookieDB Coming Back

In project 2, we implemented an Alt. 2 B+ Tree where we did not allow duplicate keys. Now let's try modifying the architecture a bit, so our B+ Tree allows inserting duplicate keys with different RecordIds. In this question, we are implementing an Alt. 3 B+ Tree.

InnerNode's structure remains the same as the project, but LeafNodes undergo some changes, specifically:

- When we serialize a leaf node, we write:
  1. the literal value 1 (1 byte) which indicates that this node is a leaf node.
  2. an indicator value (1 byte) that is 0 if the rightSibling page is the same leaf node, or 1 if the rightSibling page belongs to the next leaf node.
  3. the page id (8 bytes) of our right sibling. (or -1 if we don't have a right sibling) *Note: when the rightSibling is -1, you can assume that the indicator value will be 1.*
  4. the number of keys (4 bytes) this **page** contains
  5. The (key, numRecords, list of records) pairs themselves where each pair has the format of a. the key b. the number of records in the list c. the list of records themselves

This is a visualization of how the serialization will look like using comments:

```
//   A tree with 2 leafNodes spanning a total of 3 pages
//   ===============================================================
//   leaf0
//   page0 (pageNum=42)                    page1 (pageNum=43)
//   +-------------------------------+ | +-------------------------+
//   | nodeType = 1                  | | | nodeType = 1            |
//   | indicator = 0                 | | | indicator = 1           |
//   | rightSibling = 43             | | | rightSibling = 44       |
//   | numKeys = 3                   | | | numKeys = 1             |
//   | (key1, 2, (rid1, rid2))       | | | (key4, 2,(rid7, rid8))  |
//   | (key2, 3, (rid3, rid4, rid5)) | | +-------------------------+
//   | (key3, 1, (rid6))             | |
//   +-------------------------------+ |
//
//   leaf1
//   page0 (pageNum=44)
//   +-------------------------+
//   | nodeType = 1            |
//   | indicator = 1           |
//   | rightSibling = -1       |
//   | numKeys = 2             |
//   | (key1, 1, (rid1))       |
//   | (key2, 1, (rid2))       |
//   +-------------------------+
```

In addition, we are given the following information:

- A (key, listRecords) pair and the corresponding metadata (nodeType, indicator, rightSibling, numKeys) used to serialize/deserialize it can at most span 1 page. This means we do not need to worry about the case where a list of records corresponding to a single key spans multiple pages.
- Because each (key, listRecords) pair in a Leaf Node can span at most one page, a leaf node itself can span multiple pages (as seen in the example).
- Some useful project code information has been copied below for your convenience:

```
/* Constructor for RecordId */
public RecordId(long pageNum, short entryNum) {
   this.pageNum = pageNum;
   this.entryNum = entryNum;
}

/* Extract exact number of bytes according to the schema and convert it to RecordId */
public static RecordId fromBytes(Buffer buf) {
    return new RecordId(buf.getLong(), buf.getShort());
}

/* Implementation of LeafNode supporting list of records */
class LeafNode extends BPlusNode {

    /* The ‘‘leading’’ page of the leaf node */
    Private Page page;
    private List<DataBox> keys;
    private List<List<RecordId>> rids;

    ... // other private fields
}
```

Fill in the blanks for each function below so the Alt 3 B+ Tree with duplicate keys works as intended.

i. (12 points) fromBytes

`buf.get` will return you the next byte read, and you are also provided with all the buf methods like `buf.getInt`, `buf.getShort`, `buf.getLong` etc.

```
/**
* Loads a leaf node starting at page `pageNum`.
*/
public static LeafNode fromBytes(BPlusTreeMetadata metadata, BufferManager bufferManager,
                                 LockContext treeContext, long pageNum) {

    List<DataBox> keys = new ArrayList<>();
    // we are creating a list of list of records
    List<List<RecordId>> rids = new ArrayList<>();

    Optional<Long> rightSibling;
    byte belongsToNextNode;

    Page page = bufferManager.fetchPage(treeContext, pageNum);
    Buffer buf = page.getBuffer();

    do {
        byte nodeType = buf.get();
        assert(nodeType == (byte) 1);

        belongsToNextNode = buf.get();
        long s = __(A)__;
        rightSibling = s == -1 ? Optional.empty() : Optional.of(s);

        int n = buf.getInt();
        List<RecordId> listRecords;
        for (int i = 0; i < n; ++i) {
            keys.add(DataBox.fromBytes(buf, metadata.getKeySchema()));
            int numRecords = __(B)__;
            listRecords = new ArrayList<>();
            for ( __(C)__ ) {
                [ __(D)__ ];
            }
            [ __(E)__ ];
        }

        if (s != -1L) {
            page = bufferManager.fetchPage(treeContext, __(F)__);
            buf = page.getBuffer();
        }
    } while (__(G)__);

    Page leadingPage =  bufferManager.fetchPage(treeContext, __(H)__);

    return new LeafNode(metadata, bufferManager, leadingPage,
                            keys, rids, rightSibling, treeContext);
}
```

A. (1 pt) Fill in the code at (A).

```
buf.getLong()
```

B. (1 pt) Fill in the code at (B)

```
buf.getInt()
```

C. (2 pt) Fill in the code at (C)

```
int j = 0; j < numRecords; ++j
```

D. (2 pt) Fill in the code at (D)

```
listRecords.add(RecordId.fromBytes(buf)) or listRecords.add(fromBytes(buf))
```

E. (2 pt) Fill in the code at (E)

```
rids.add(listRecords)
```

F. (1 pt) Fill in the code at (F)

```
s
```

G. (2 pt) Fill in the code at (G)

```
belongsToNextNode == 0
```

H. (1 pt) Fill in the code at (H)

```
pageNum
```

ii. (8 points) put

Now let's also modify the `LeafNode.put` method as it allows duplicate keys! You should modify **keys** and **rids** so it reflects the addition.

```
public Optional<Pair<DataBox, Long>> put(DataBox key, RecordId rid) {
    int index = InnerNode.numLessThan(key, keys);
    List<RecordId> listRecords;
    if (__(A)__) {
        [ __(B)__ ]
        listRecords.add(rid);
    } else {
        [ __(C)__ ];
        listRecords = new ArrayList<>();
        listRecords.add(rid);
        [ __(D)__ ];

        /* Where we used to have normal key insertion */
        if (keys.size() <= 2 * metadata.getOrder()) {
            ...
        } else {
            ...
       }
    }
}
```

A. (2 pt) Fill in the code for (A)

`keys.contains(key) Alternative solution:  keys.get(index) == key`

B. (2 pt) Fill in the code for (B)

`listRecords = rids.get(index);`

C. (2 pt) Fill in the code for (C)

`keys.add(index, key)`

D. (2 pt) Fill in the code for (D)

`rids.add(index, listRecords)`

No more questions.