CS 186 Fall 2020 Midterm 1 (Online)

Do not share this exam until solutions are released.

Contents:

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

Taking the exam:

- You have *110 minutes* to complete the midterm.
- You may print this exam to work on it.
- For each question, submit only your *final answer* on examtool.
- For numerical answers, do not input any suffixes (i.e. if your answer is 5 I/Os, only input 5 and not 5 I/Os or 5 IOs) and do not use LaTeX.
- Some questions require you to show work and not doing so will result in **no credit**. You can do this by inputting an explanation in the text field. The text field supports LaTeX by using $$insert expression here$$ but don't worry too much about formatting.
- Make sure to save your answers in examtool at the end of the exam, although the website should autosave your answers as well.

Aids:

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

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.

1 Pre-Exam Question (0 points)

When was the last time CS186 lectures were taught in person? Give us the season and year (ex. Fall 2020).

2 More Baseball! (20 points)

In this problem we will use a simplified version of the database we used in project 1:

```
CREATE TABLE Player (
    pid INTEGER PRIMARY KEY,
    age INTEGER NOT NULL,
    name VARCHAR[30] NOT NULL,
    salary INTEGER NOT NULL,
    rating INTEGER NOT NULL
);
CREATE TABLE Team (
    tid INTEGER PRIMARY KEY,
    name VARCHAR[30] NOT NULL,
);
CREATE TABLE MemberOf (
    pid INTEGER PRIMARY KEY,
    tid INTEGER NOT NULL,
    FOREIGN KEY (pid) REFERENCES Player
    FOREIGN KEY (tid) REFERENCES Team
);
```

You can assume none of the tables contains NULL values.

1. (4 points) Given the following contents of Player:

pid	name	age	salary	rating
1	"P1"	30	50000	3
2	"P2"	24	10000	3
3	"P3"	21	15000	2

   Write out the result for the following query in a comma-delimited form, where the first row represents the column names. For instance, if the query returned the following result:

pid	name	age	salary	rating
1	"P1"	30	50000	3

   Express it as:

   pid, name, age, salary, rating
   1, "P1", 30, 50000, 3

```
SELECT P1.rating AS rating, AVG(P1.age) AS avgAge
FROM Player P1
WHERE P1.age > 21
GROUP BY P1.rating
HAVING 1 < (SELECT COUNT(*)
            FROM Player P2
            WHERE P1.rating = P2.rating AND P2.age >= 21)
```

   Your answer:

Using the same schema as given previously, follow these instructions for the following parts.

Does each of the following query pairs always return the same results regardless of the contents of the involved tables? If so, write "Yes" below. Otherwise, write "No," and make up the contents of `Player` such that, when run with the two queries, they will return different results. You do not need to include any explanation for either answer.

For "No" answers, write the table contents in the comma-delimited form as in the first question. Again, if your answer is the following for the `Player` table:

Player

pid	name	age	salary	rating
1	"P1"	30	50000	3

Express it as:

Player:
pid, name, age, salary, rating
1, "P1", 30, 50000, 3

Note that some queries involve multiple tables so be sure to provide table names as in the above. Also, make sure your tuples satisfy all key constraints!

2. (4 points) Q1:

```
SELECT P1.name
FROM Player P1
WHERE NOT EXISTS (SELECT *
                  FROM Player P2
                  WHERE P2.age < 21 AND P1.rating <= P2.rating)
```

Q2:

```
SELECT P1.name
FROM Player P1
WHERE P1.rating > ANY (SELECT P2.rating
                       FROM Player P2
                       WHERE P2.age < 21)
```

Your answer:

3. (4 points) Q1: $\sigma_{\texttt{salary<10k}}(\sigma_{\texttt{age<21}}(\texttt{Player}) \bowtie_{\texttt{Player.pid=MemberOf.pid}} (\sigma_{\texttt{tid>40}}(\texttt{MemberOf})))$
   Q2: $\sigma_{\texttt{salary<10k}}(\texttt{Player} \bowtie_{\texttt{Player.pid=MemberOf.pid}} (\sigma_{\texttt{tid>40}}(\texttt{MemberOf})))$

   Your answer:

4. (4 points) Q1: $\gamma_{\texttt{Team.tid, max(salary)}}((\texttt{Player} \bowtie_{\texttt{Player.pid=MemberOf.pid}} \texttt{MemberOf}) \bowtie_{\texttt{Memberof.tid=Team.tid}} \texttt{Team})$
   Q2: $\gamma_{\texttt{Team.tid, max(salary)}}(\texttt{Player} \bowtie_{\texttt{Player.pid=MemberOf.pid}} \texttt{MemberOf})$

   Your answer:

5. (4 points) Which of the following returns the same results as the following relational algebra expression? Select all queries that return the same results. No explanations needed. There may be 0, 1, or more correct answers.

$\rho_{\texttt{rating->r, avg(salary)->sal}}(\gamma_{\texttt{rating, avg(salary)}}(\texttt{Player}))$

◯ Query A

```
SELECT P2.rating as r, AVG(P2.salary) as sal
FROM Player AS P2,
     (SELECT DISTINCT P1.salary AS salary
      FROM Player AS P1
      WHERE P1.rating = P2.rating) AS t
WHERE P2.rating = t.rating
```

◯ Query B

```
SELECT DISTINCT P2.rating AS r,
       (SELECT AVG(t.salary)
        FROM (SELECT P1.salary AS salary
              FROM Player AS P1
              WHERE P1.rating = P2.rating) AS t) AS sal
FROM Player AS P2
```

◯ Query C

```
SELECT P2.rating AS r, AVG(P2.sal1) AS sal
FROM (SELECT P1.rating, P1.age, AVG(P1.salary) AS sal1
      FROM Player AS P1
      GROUP BY P1.rating, P1.age) AS P2
GROUP BY P2.rating
```

◯ Query D

```
SELECT P2.rating AS r,
       (SELECT AVG(t.salary)
        FROM (SELECT P1.salary AS salary
              FROM Player AS P1
              WHERE P1.rating = P2.rating) AS t) AS sal
FROM Player AS P2
```

3 Where's the Zoom Record(ing)? (25 points)

Zoom cloud recording has became too expensive for the university, so the CS 186 TAs have been hired to design a database to store lecture recording information. The TAs need your help in considering different designs!

For the rest of the questions consider the following schema:

```
CREATE TABLE zoomers (
    lecture_id INTEGER PRIMARY KEY,
    week INTEGER NOT NULL,
    topic VARCHAR[30] NOT NULL,
    num_thanks INTEGER,
    poppin_chat BOOLEAN
);
```

For questions 1-5, assume the following:

- `zoomers` is stored in a Heap File Page Directory implementation. There are 25 data pages and each header page has 6 entries.
- Data pages follow the slotted page layout as presented in lecture.
- On each data page, 8 bytes are reserved for the slot count and pointer to free space. Each slot is 8 bytes.
- Records are stored as variable length records
- The record header contains a bitmap to track all fields that can be `NULL`. The bitmap is as small as possible, rounded up to the nearest byte.
- Integers and pointers are 4 bytes. Booleans are 1 byte.
- Each page is 1 KB (1024 Bytes).
- There are no indices on any field.

1. (2 points) What is the maximum size of a `zoomers` record in bytes?

2. (2 points) What is the maximum number of `zoomers` records that can fit on a data page?

For questions 3-5, assume the queries are independent of each other; i.e. query 4 is run on a copy of the file that has never had query 3 run on it. The buffer is large enough to hold all data and header pages, and starts empty **for each question**.

3. (3 points) What is the **worst** case cost in I/Os for the following query?

```
DELETE FROM zoomers WHERE poppin_chat = FALSE;
```

4. (3 points) What is the **best** case cost in I/Os for the following query?

```
SELECT num_thanks FROM zoomers WHERE lecture_id = 186;
```

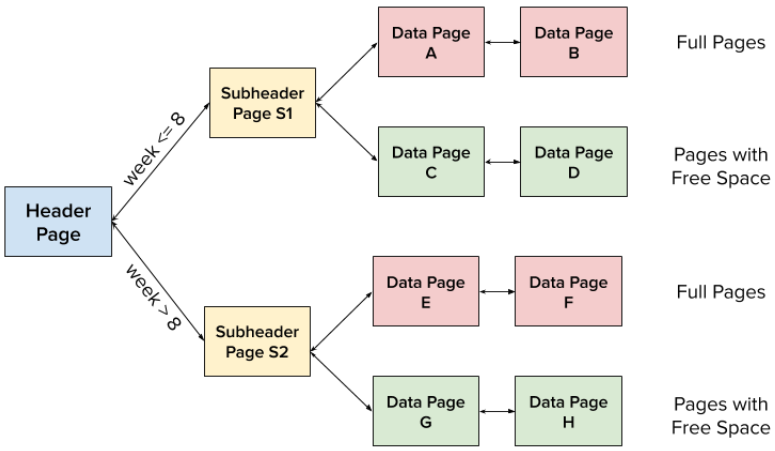
5. (4 points) What is the **best** case cost in I/Os for the following query?

```
SELECT week, topic FROM zoomers WHERE lecture_id > 100 AND lecture_id <= 286;
```

Thinking that students will access lectures based on the `week`, you have the brilliant idea of storing the `zoomers` table as a sorted file based on the `week` column. Unfortunately, The Great California Fires of 2020 obliterated some Zoom datacenters, and the lecture recording on sorted files is lost forever! Without that information you decide to come up with the following design for a partitioned linked list (PLL) heap file:

The header page in the heap file linked list implementation now points to P subheader pages. Each subheader points to a list of full pages and a list of pages with free space. Each subheader and its 2 lists are collectively called a partition; hence there are P partitions. Each record in the file is placed in a partition based on the `week`.

An example PLL heap file with 2 partitions is shown below with the assumption that $week >= 1$ and $week <= 16$. That assumption cannot be made for questions 6-8.



For questions 6-8 assume the following:

> https://www.overleaf.com/project/5f3c264872166c00013e713fThe header page stores 8 byte subheader entries. Each subheader entry is a pair of `week` and a pointer to the corresponding subheader page. P = number of partitions N = number of data pages Each partition has $\frac{N}{P}$ data pages. Assume $\frac{N}{P}$ will always be an integer. Each partition has an equal number of full pages and pages with free space. There are at least 3 pages with free space (and 3 full pages) in each partition The buffer is large enough to hold all data and header pages, and starts empty **for each question**. You can use `ceil(x,y)` to represent $\lceil \frac{x}{y} \rceil$

6. (4 points) What is the worst case I/O cost for inserting a `zoomers` record in terms of P and N?

To account for multiple courses sharing the same Zoom account to record lectures, let's add a field for `course_id` to the table and modify the **`PRIMARY KEY`**. The new `zoomers` schema is:

```
CREATE TABLE zoomers (
    lecture_id INTEGER,
    week INTEGER NOT NULL,
    topic VARCHAR[30] NOT NULL,
    num_thanks INTEGER,
    poppin_chat BOOLEAN,
    course_id INTEGER,
    PRIMARY KEY(course_id, lecture_id)
);
```

For questions 7-8, let C = number of courses and assume there are now N data pages **for each course**. Partitions are still based on `week` and each partition now has $\frac{CN}{P}$ data pages, which you can assume is an integer.

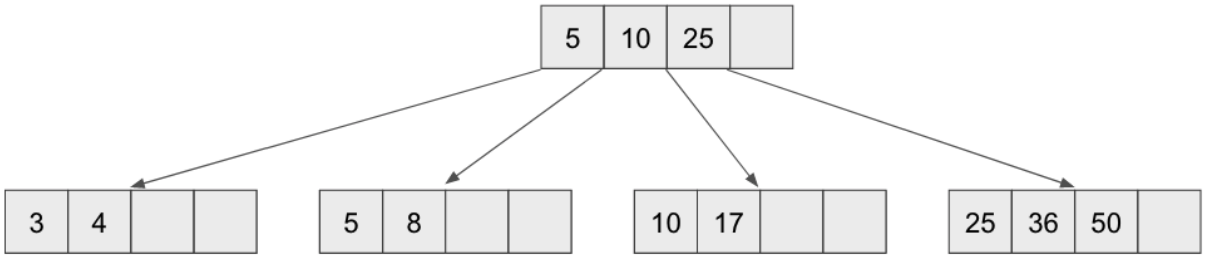
7. (3 points) What is the worst case cost in I/Os to perform an equality search on a given `week` and `course_id`? Answer in terms of P, N, and C.

8. (4 points) In 2-3 sentences, propose a change to the PLL heap file that minimizes the worst case I/O cost to to perform an equality search on a given `week` and `course_id`. Provide the worst case cost in I/Os in terms of P, N, and C? Your proposal may not use indexes or sorted files.

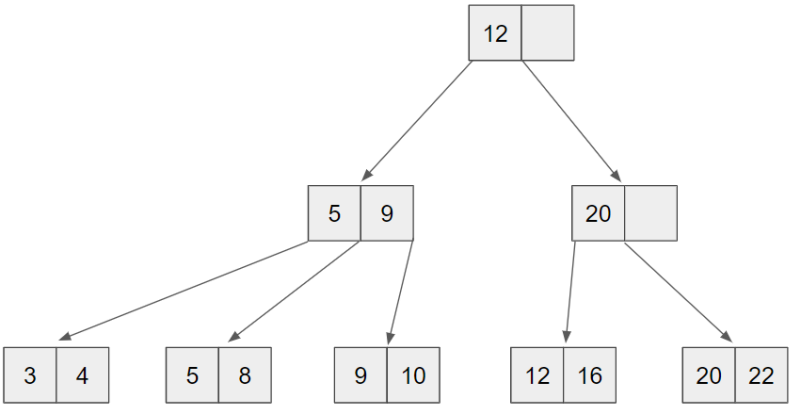
4 B+ Trees (25 points)

1. (2 points) Which of the following are always true regarding B+ Trees? **There may be zero, one, or more than one correct answer.**
   A. If $d = 2$, then the root node must have at least two entries.
   B. Suppose we maintain both an Alternative 2 and an Alternative 3 B+ Tree on the primary key of some table. Inserting a value into the Alternative 2 B+ Tree will cost the same number of I/Os as inserting into the Alternative 3 B+ Tree. Assume a leaf node is stored on exactly one page.
   C. Suppose we have two entries A and B that we want to insert into a B+ Tree. Inserting A then B into the B+ Tree results in the same final B+ Tree as inserting B then A.
   D. Suppose we want to bulk load a B+ Tree with entries 1 to 10 with a fill factor of $\frac{3}{4}$. The B+ Tree has an order of 2. After bulk loading with 1 to 5, the leftmost leaf node is never touched again.
2. (2 points) Suppose we have a different type of B+ Tree where leaf nodes and inner nodes don't necessarily have the same order. If we have $d = 2$ for inner nodes and $d' = 3$ for leaf nodes, what is the maximum number of records we can store in an Alternative 2 B+ Tree of height 3?

For question 3 and 4, suppose we have the following B+ Tree with order $d = 2$.



3. (2 points) What is the minimum number of records we can insert to increase the height of the tree?
4. (3 points) After inserting entries 51 to 100 (inclusive) one by one into the tree, what values are in the rightmost leaf node? Hint: Try inserting a few numbers and see if you can find a pattern.
5. (3 points) Suppose we want to bulk load a B+ Tree with order $d = 1$. We want to bulk load 100 records using a fill factor of 1. Is the following B+ Tree a possible intermediate step during our bulk load process? Explain your answer.



For questions 6- 10, suppose we have the following table shown below. Also, assume that our buffer is large enough to hold all pages needed and is empty at the start of each question.

```
CREATE TABLE Restaurants (
    rid INTEGER PRIMARY KEY,
    name VARCHAR,
    type VARCHAR,
    size INTEGER,
    num_ratings INTEGER,
    rating FLOAT
);
```

6. (2 points) Suppose we have a height 3 Alternative 1 B+ Tree with order $d = 2$ on `rid`. What is the minimum number of I/Os it would take to execute the following query (Assume there is at least one matching tuple):

```
UPDATE Restaurants SET num_ratings = 500 WHERE rid = 100;
```

7. (2 points) Now suppose we have an unclustered Alternative 2 B+ Tree instead of an Alternative 1 B+ Tree. What is the minimum number of I/Os it would take to execute the same query (Assume there is at least one matching tuple)?

We decide to mix things up by creating an Alternative 2 unclustered B+ Tree with height 3 and order $d = 2$ on (size, num_ratings). As a reminder, for multi-column keys, comparisons are done of the first element in the key, and the second elements are compared to break ties. For problems 8 -10, assume the following:

- There are no duplicate (size, num_ratings) pairs.
- There are exactly 5 records per page in the heap file corresponding to the B+ Tree.
- There are 100 leaf nodes with exactly 3 entries in each leaf node. Each leaf node fits exactly on one page.
- The table has 60 data pages in total.
- The first entry with size=20 appears on the first entry of the 91st leaf node.
- The last entry with num_ratings=500 appears on the last entry of the 86th leaf node.
- None of the index pages are in the buffer pool at the start of each part.

Now, suppose we want to execute the following query:

```
SELECT * FROM Restaurants WHERE size >= 20 AND num_ratings <= 500;
```

8. (3 points) In the worst case what is the number of I/Os it would take to execute the query?
9. (3 points) If we had a clustered index instead of an unclustered index, how many I/Os would it take to execute the query in the worst case scenario?

   Now, assume we want to execute the following query:

```
SELECT * FROM Restaurants WHERE size >= 20 OR num_ratings <= 500;
```

10. (3 points) What is the minimum number of I/Os it would take to execute the query? Assume that the index is unclustered for this part.

5 Buffer Management (10 points)

1. (2 points) Which of the following statements are true? **There may be zero, one, or more than one correct answer.**
    A. If a page is dirty then it must have at least one pin.
    B. The clock algorithm is used in favor of LRU because it isn't susceptible to sequential flooding.
    C. The buffer manager is only allowed to evict a pinned page if all other pages are also pinned.
    D. If the dirty bit of a page is set the buffer manager should write the page back to disk upon eviction.

For questions 2 and 3 assume you have 4 buffer frames, all accesses are unpinned immediately, and empty frames are filled in order. Consider the following access pattern:

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

2. (1 point) Which algorithm would be better for this access pattern, LRU or MRU?
3. (1 point) Explain your answer to the previous question.

4. (2 points) Consider the Least Frequently Used (LFU) eviction policy. Whenever a page is loaded into the buffer a 32-bit value representing the page's hit rate is set to 1. Every time the page is requested afterwards the hit rate is incremented. When a page needs to be evicted, this policy chooses the page with the lowest hit rate (breaking ties randomly). Consider the following workload: for one day every month a specific index is used heavily and continuously to perform analysis of a table. The remainder of the month the table is only used occasionally and work is spread out across other tables. Describe why an LFU policy might be problematic here. Assume that the counters never overflow.

5. (2 points) Describe an adjustment to the LFU policy that can help to mitigate the problem you described in the previous question. There are many possible correct answers to this question, just make sure to retain some aspect of the original policy (i.e. don't say "Just use LRU instead").

6. (2 points) In lecture we mentioned that neural networks can be used to choose which page we should evict next. Imagine that recent breakthroughs in machine learning and Big Data™ have given you access to a neural network that always predicts the OPT policy correctly. The OPT policy chooses a page to evict such that cache hits are maximized and cache misses minimized; it evicts optimally for all access patterns but take awhile to process all that Big Data™. Briefly explain a situation where you wouldn't want to use this method for buffer management.

6 Sorting and Hashing (20 points)

Suppose we are trying to do external sorting or hashing on the following relation:

```
CREATE TABLE Students (
    calid INTEGER PRIMARY KEY,
    zipcode VARCHAR NOT NULL,
    department VARCHAR NOT NULL
);
```

This table contains information about 500,000 students, split on disk across 1000 pages. `department` can take on one of 10 distinct values; `zipcode` can be one of 1000 values. You can assume a uniform distribution across departments, i.e., the same number of students are part of each department, and zipcodes, i.e., the same number of students are in each zipcode. We have $B = 20$ pages in our DBMS buffer.

A calculator is recommended for this question.

1. (5 points) Say we wanted to sort the tuples in `Students` based on the `zipcode`.

   How many I/Os would we need for this algorithm?

   How many *runs* would you read into memory across all phases of the algorithm? (Recall that a run is a sorted sequence of blocks. We're not asking about the length of the run, just the number of runs. Do not treat the initial reading-in of the table as reading in any runs.)

2. (5 points) Say we wanted to sort the tuples in `Students` based on the `zipcode` once again.

   This time, you are going to derive an improved version of the previous algorithm on the fly! The key insight for this improvement is that we did not fully utilize all of the B pages in the buffer pool during every stage of the algorithm. Briefly explain how you would improve the previous algorithm using this insight. How many I/O units would we need using our improved algorithm?

   Hint: you could use some of the unused pages in the buffer to "start early" on the next phase of the algorithm.

3. (5 points) Say we wanted to hash the tuples in `Students` based on the `zipcode`.

   How many I/O units would we need for this algorithm?

   You can assume that we use a perfect hash function that divides values up evenly.

4. (5 points) Say we wanted to hash the tuples in `Students` based on the `zipcode` once again.

   Let's once again try to develop a better algorithm for this on the fly! For designing this algorithm, say we have more information: one of the `zipcodes` (e.g., the Berkeley 94705 zipcode) is really popular—with $\frac{3}{5}^{th}$ of the students living in this `zipcode`, while the rest of the students are evenly distributed across the remaining `zipcode` s. Moreover, we can make use of this information in our hash function, i.e., you can design the ideal hash function you want. Briefly explain how you could improve the algorithm using this insight.

   How many I/Os would we need for this algorithm?

   Hint: Consider using a hash function that treats 94705 as a special value.

7 Scratch Work (0 points)