CS W186 Spring 2020 Midterm 1

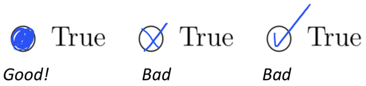
Do not turn this page until instructed to start the exam.

Contents:

- You should receive one *double-sided answer sheet* and a 15-page *exam packet.*
- The midterm has *6 questions*, each with multiple parts, and worth a total of *84 points*.

Taking the exam:

- You have *110 minutes* to complete the midterm.
- All answers should be written on the answer sheet. The exam packet will be collected but not graded.
- For each question, place only your *final answer* on the answer sheet; *do not show work.*
- For multiple choice questions, please *fill in the bubble or box completely* as shown on the left below. ***Answers marking the bubble or box with an X or check mark may receive a point penalty***.



- Bubbles indicate only one answer while boxes indicate one or multiple answers.
- A blank page is provided at the end of the exam packet for use as scratch paper.

Aids:

- You are allowed **one handwritten** 8.5" × 11" double-sided pages of notes.
- *No electronic devices are allowed on this exam.* No calculators, tablets, phones, smartwatches, etc.

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)

1. (0.0001 points) Write down the names of all CS W186 faculty and TAs whose first name starts with 'J' (hint: there are 9).

2 Octograder Queries (18 points)

It is the beginning of a new semester and the CS W186 TAs are setting up their new autograder, the Octograder. They find you, who is a master at SQL, to help them set up the student roster in Octograder. The goal is to match the students enrolled in the CS W186 edX to the students enrolled in this class on CalCentral using hw0. The roster downloaded from the edX course page, the course roster on CalCentral, and the hw0 submission responses downloaded from edX are already uploaded into Octograder as tables, with the following schemas:

```
CREATE TABLE edx_students (
    edx_id INTEGER PRIMARY KEY,
    name VARCHAR,
    email VARCHAR UNIQUE
);

CREATE TABLE calcentral_roster (
    user_id INTEGER PRIMARY KEY,
    name_first VARCHAR,
    name_last VARCHAR,
    sid INTEGER UNIQUE,
    email VARCHAR UNIQUE
);

CREATE TABLE hw0 (
    submission_id INTEGER PRIMARY KEY,
    submission_time TIMESTAMP,
    name VARCHAR,
    berkeley_email VARCHAR,
    edx_email VARCHAR,
    sid INTEGER,
    submission_file VARCHAR
);
```

Assume the following:

- `calcentral_roster` contains all students who are currently enrolled in the class. It does not contain students who dropped the class.
- `hw0` contains all submissions, including those from the students who dropped the class.
- All `sid`s and `berkeley_email`s in `hw0` are correct. There are no typos in these fields.
- Students can submit hw0 multiple times, because they might have entered the wrong `edx_email`. All students who submitted hw0 have the correct `edx_email` in their latest submissions.
- `berkeley_email` and `edx_email` may or may not be the same.

Before you start, the head TA tells you that there is a student who added the class late and might not show up in `calcentral_roster`, which would cause problems later in the semester. So you want to find if this student is on the roster. His first name is 'Lakya' and last name starts with 'J'. You write the following query to find all such students.

```
SELECT *
FROM calcentral_roster
WHERE _ _ (1) _ _
AND _ _ (2) _ _
```

1. (1.5 points) Select all possible options for blank 1, which finds all first names that are 'Lakya'.

   A. `name_first = ‘Lakya’`

   B. `name_first ~ ‘Lakya’`

   C. `name_first LIKE ‘Lakya’`

   D. None of the above

2. (1.5 points) Select all possible options for blank 2, which finds all last names that start with 'J'.

   A. `name_last = ‘J.*’`

   B. `name_last ~ ‘J.*’`

   C. `name_last LIKE ‘J%’`

   D. None of the above

3. (3 points) You want to email all the enrolled students who did not submit hw0 and remind them that they will be administratively dropped if they do not submit hw0.

   Select the `berkeley_email` of all enrolled students who did not submit hw0.

   There may be zero, one, or multiple correct answers.

   A.
   ```
   SELECT C.email
   FROM hw0 H FULL OUTER JOIN calcentral_roster C
   ON H.sid = C.sid
   WHERE H.sid IS NULL;
   ```

   B.
   ```
   SELECT email
   FROM calcentral_roster
   WHERE email NOT IN (
       SELECT berkeley_email
       FROM hw0);
   ```

   C.
   ```
   SELECT C.email
   FROM (
       SELECT sid
       FROM hw0
   ) AS H RIGHT OUTER JOIN calcentral_roster C
   ON H.sid = C.sid
   WHERE C.sid IS NULL;
   ```

   D. None of the above

4. (3 points) Before you start matching students, you want to create a table with only the latest hw0 submissions from every student, because you only want to use the `edx_email` in everyone's latest submission.

   Select the `name`, `berkeley_email`, `edx_email`, `sid`, `submission_file` for each student's latest hw0 submission.

   There may be zero, one, or multiple correct answers.

   A.
   ```
   SELECT name, berkeley_email, edx_email, sid, submission_file
   FROM hw0 H
   WHERE H.submission_time >= ALL (
       SELECT H1.submission_time
       FROM hw0 H1
       WHERE H1.sid = H.sid);
   ```

   B.
   ```
   SELECT name, berkeley_email, edx_email, sid, submission_file
   FROM hw0 H
   GROUP BY sid
   HAVING H.submission_time >= ALL (
       SELECT H1.submission_time
       FROM hw0 H1
       WHERE H1.sid = H.sid);
   ```

   C.
   ```
   SELECT name, berkeley_email, edx_email, sid, submission_file
   FROM hw0 H INNER JOIN (
       SELECT sid, MAX(submission_time) as latest_hw0_time
       FROM hw0
       GROUP BY sid) AS H1
   ON H.submission_time = H1.latest_hw0_time
   AND H.sid = H1.sid;
   ```

   D. None of the above

Assume the table described in the previous question is correctly created, so now there is a table called `latest_hw0` with columns `name`, `berkeley_email`, `edx_email`, `sid`, `submission_file` and only contains the hw0 submissions that are the latest submissions for each student who submitted.

5. (3 points) Now you are ready to match students. Select the `sid` and `edx_id` of all the enrolled students. The output should have one row for every enrolled student, and only enrolled students are in the output. If a student did not submit hw0, the `edx_id` should be **`NULL`**.

   There may be zero, one, or multiple correct answers.

   A.
   ```
   SELECT C.sid, E.edx_id
   FROM calcentral_roster C, edx_students E, latest_hw0 L
   WHERE C.sid = L.sid
   AND E.email = L.edx_email
   AND C.sid IS NOT NULL;
   ```

   B.
   ```
   SELECT C.sid, E.edx_id
   FROM calcentral_roster C
   LEFT OUTER JOIN latest_hw0 L ON C.sid = L.sid
   LEFT OUTER JOIN edx_students E on E.email = L.edx_email;
   ```

   C.
   ```
   SELECT C.sid, E.edx_id
   FROM edx_students E
   FULL OUTER JOIN latest_hw0 L on E.email = L.edx_email
   RIGHT OUTER JOIN calcentral_roster C on C.sid = L.sid;
   ```

   D. None of the above

For the following questions, we refer to the `edx_students` table as E, `calcentral_roster` table as C, `hw0` table as H, and `latest_hw0` table as L.

6. (3 points) Choose the relational algebra expressions that find the `sid` of all students who did not submit hw0.

   There may be zero, one, or multiple correct answers.

   A. $\pi_{\text{sid}}(C) - \pi_{\text{sid}}(H)$

   B. $\pi_{\text{sid}}(C) - \pi_{\text{sid}}(L)$

   C. $\pi_{\text{sid}}(C) - \pi_{\text{sid}}(C \bowtie_{\text{C.sid = L.sid}} (L))$

   D. None of the above

7. (3 points) Choose the relational algebra expressions that find all the `edx_id` of students whose `berkeley_email` is different from their `edx_email`.

   There may be zero, one, or multiple correct answers.

   A. $\pi_{\text{edx\_id}}(E) - \pi_{\text{edx\_id}}(\sigma_{\text{berkeley\_email=edx\_email}}(L))$

   B. $\pi_{\text{edx\_id}}(E) - \pi_{\text{edx\_id}}(E \bowtie_{\text{email=edx\_email}} (\pi_{\text{edx\_email}}(L)))$

   C. $\pi_{\text{edx\_id}}(\sigma_{\text{edx\_email<>berkeley\_email}}(L) \bowtie_{\text{edx\_email=email}} (\pi_{\text{email}}(E)))$

   D. None of the above

3 Big Disk Energy (23 points)

1. (4 points) Which of the following statements are true? **There may be zero, one, or more than one correct answer.**

   A. The slotted page design is usually used for pages with fixed length records.

   B. In the best case, the page directory implementation of a heap file always costs at most the same number of I/Os as the linked list implementation for checking if there exists a page with enough empty space.

   C. In the slotted page implementation, the pointers in the footer all point to the end of the records.

   D. In variable length records, the length of each field is stored in the header.

Welcome to Silicon Valley! You are a fresh graduate from UC Berkeley, the number one public university, and you were just hired by Flux Motors as their database engineer.

Alon Tusk is way too busy making memes and making Flux stock skyrocket to the moon! He tasks you with optimizing Flux's database where car orders are kept.

For the following questions, consider the following schema:

```
CREATE TABLE flux_orders (
    orderID INTEGER PRIMARY KEY,
    name VARCHAR[20] NOT NULL,
    email VARCHAR[20] NOT NULL,
    model INTEGER NOT NULL
);
```

You may assume the following for the problems below:

- Integers and pointers are 4 bytes each.
- 1 KB = 1024 bytes
- Page headers contain only an integer for the record count.
- Slot directories are implemented as seen in lecture and notes.

2. (2 points) You find out that Flux is using fixed length records for their car orders. What is the maximum number of records you can fit on a 1KB page **using fixed length records**?

3. (3 points) After having taken CS W186, you suggest using variable length records instead. What is the maximum number of records you can fit on a 1KB page **using variable length records**?

You remember learning about two types of heap file implementations. You decide to investigate the I/O costs for different queries for these two implementations.

- Assume there are 18 data pages and page directory headers can hold pointers to 10 data pages.
- Assume there are 12 full pages and 6 pages with free space in the linked list implementation
- Assume the buffer is big enough to hold all data and header pages and starts empty for each question/part.
- There are no indices on any field.
- Assume all queries are independent of each other; i.e. query 2 is run on a copy of the file that has never had query 1 run on it.

For questions 4 and 5, calculate the number of I/Os it would take to complete the given query in the **worst case** when using the **Page Directory implementation**.

4. (2 points) `SELECT * FROM flux_orders WHERE orderID > 99 AND orderID < 1000`

5. (3 points) `INSERT INTO flux_orders VALUES (1, "Oski", "oski@berkeley.edu", 3)`

For questions 6 and 7, calculate the number of I/Os it would take to complete the given query in the **worst case** when using a **Linked List implementation**.

6. (2 points) `SELECT * FROM flux_orders WHERE orderID > 99 AND orderID < 1000`

7. (3 points) `INSERT INTO flux_orders VALUES (1, "Oski", "oski@berkeley.edu", 3)`
   - Assume after insertion, the data page **does not** become full.

8. (4 points) How many **additional** I/Os from your answer in question 7 are required if after insertion, the page **does** become full?

4 A+ Trees (15 points)

1. (1 point) Which **index(es) on name** would help to speed up the following query given that many users share the same name (select all that apply)

   INSERT INTO users (name)
   VALUES (David);

   A. No index

   B. Alt 1 Index

   C. Alt 2 Clustered Index

   D. Alt 2 Unclustered Index

   E. Alt 3 Clustered Index

   F. Alt 3 Unclustered Index

2. (1 point) Given that grades are from [1-12] with 50 students per grade, which **index on grade would help to speed up the following query the most**

   SELECT * FROM students WHERE grade >= 10;

   A. No index

   B. Alt 1 Index

   C. Alt 2 Clustered Index

   D. Alt 2 Unclustered Index

   E. Alt 3 Clustered Index

   F. Alt 3 Unclustered Index

3. (1 point) Given a B+ Tree containing values [1-10], after inserting values [100-150], the height of the root's leftmost subtree will be much greater than the height of all the other root's subtrees.

   A. True

   B. False

For questions 4-8, use the following table and indices to answer the questions. Assume all queries are independent of each other.

```
CREATE TABLE colleges {
    name varchar[255] PRIMARY KEY,
    ranking int,
    city varchar[255],
    state varchar[255],
    size int,
    happiness int
}
```

The **colleges** table is stored as a **heap file** and is **implemented using a Page Directory** containing **5 header pages** and **100 data pages**. There is a **height 3 Alt 2 unclustered index on ranking** and a **height 4 Alt 2 unclustered index on size**. All indices are of order **d = 2**.

4. (2 points) What is the **worst case** I/O cost to execute the following query:

   SELECT * FROM colleges WHERE name = ’UCLA’;

5. (2 points) In the **worst case**, what is the I/O cost to perform an **index scan on ranking** assuming that **memory is not large enough to hold all pages** and that the **index is completely full** (all nodes are completely filled and there are no missing nodes)?

For questions 6-8, creating new nodes costs 1 I/O as a new page is allocated in memory and written to disk after new values are inserted.

6. (4 points) In the **worst case**, what is the I/O Cost to execute the following query:

   INSERT INTO colleges (name, ranking, city, state, size, happiness)
   VALUES (UC Berkeley, 1, Berkeley, CA, 36000, 10);

   Assume that there is space in the heap file to insert a new record and that there is no need to check for duplicate primary keys.

7. (3 points) To optimize equality queries on happiness score, we decide to build an order **d=1 Alt 2 index on the happiness column using bulkloading with a leaf node fill factor of 1**. If the buffer pool contains 3 pages and uses LRU eviction, what is the total I/O cost of bulkloading the values into an empty index:

   3, 8, 10, 11, 1, 4

   Assume that the **values are already in memory** and do not take up space in the 3 buffer pool pages.

8. (1 point) Given a table containing 95% of all student's exam scores, what fill factor for leaf nodes would be best for bulkloading a B+ Tree knowing that the staff will use the index to perform many range queries on the scores and periodically insert the remaining 5% of scores.
   - A. 1/4
   - B. 1/2
   - C. 3/4
   - D. 1

5 Assistant to the Buffer Manager (12 points)

1. (4 points) Which of the following statements are true? **There may be zero, one, or more than one correct answer.**
   A. It is the buffer manager's responsibility to decide when a page should be unpinned.
   B. Sequential flooding is especially prone to occur when the same page is read many times consecutively.
   C. The reference bit in the clock policy serves as a replacement for the pin count that the other replacement policies require.
   D. If a database had the same amount of RAM as it did disk space, there would be no need to implement a replacement policy in the buffer manager.

For 2-11, assume you have 4 buffer frames, MRU and clock are independent, all accesses are unpinned immediately, empty frames are filled in order, and the clock hand starts at frame 1. You have the following workload:

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

2. (0.5 points) For MRU, was the final E a hit?
3. (0.5 points) For MRU, was the final B a hit?
4. (0.5 points) For MRU, was the final A a hit?
5. (0.5 points) For MRU, was the final D a hit?
6. (0.5 points) What is the hit rate for MRU?

7. (0.5 points) For clock, what letter is in frame 1 at the end?
8. (0.5 points) For clock, what letter is in frame 2 at the end?
9. (0.5 points) For clock, what letter is in frame 3 at the end?
10. (0.5 points) For clock, what letter is in frame 4 at the end?
11. (0.5 points) What is the hit rate for clock?

We again have four buffer frames and are now given the following workload:

A (remains pinned), B (remains pinned), C, D, E, F, (A is unpinned), H, G, (B is unpinned), E

Note that now we assume the first access to A is unpinned in between the accesses to F and H and the first access to B is unpinned in between the accesses to G and E. All other accesses are unpinned immediately and we again have four buffer frames.

12. (0.75 points) For LRU, what letter is in frame 1 at the end?

13. (0.75 points) For LRU, what letter is in frame 2 at the end?

14. (0.75 points) For LRU, what letter is in frame 3 at the end?

15. (0.75 points) For LRU, what letter is in frame 4 at the end?

6 Let's Sort/Hash This Out (16 points)

Some friends in Iowa are having trouble figuring out the exact steps for sorting a table in their database. Suppose their table is 100 pages and there are 5 buffer pages in memory. Fill out the following pass-by-pass guide which details the number and length of runs created after each pass in External Merge Sort.

In the second blank for each pass, write all unique run lengths in **increasing** order, separated by commas. In the first blank for each pass, write the number of runs corresponding to each run length specified in the second blank, separated by commas.

For example, if there were 3 runs of length 50 created after Pass 0, then write 3 for the first blank and 50 for the second blank on the line for Pass 0.

If instead there were 6 runs created in total: 2 runs of length 10, 1 run of length 25, and 3 runs of length 50, then write 2, 1, 3 for the first blank and 10, 25, 50 for the second blank

If the algorithm terminates before the last pass, write 0 for all blanks in unused passes.

- Pass 0: ____i____ run(s) of ____ii____ page(s) are created.
- Pass 1: ____iii____ run(s) of ____iv____ page(s) are created.
- Pass 2: ____v____ run(s) of ____vi____ page(s) are created.
- Pass 3: ____vii____ run(s) of ____viii____ page(s) are created.
- Pass 4: ____ix____ run(s) of ____x____ page(s) are created.

1. (0.5 points) Write down the number(s) for blank i.
2. (0.5 points) Write down the number(s) for blank ii.

3. (0.5 points) Write down the number(s) for blank iii.
4. (0.5 points) Write down the number(s) for blank iv.

5. (0.5 points) Write down the number(s) for blank v.
6. (0.5 points) Write down the number(s) for blank vi.

7. (0.5 points) Write down the number(s) for blank vii.
8. (0.5 points) Write down the number(s) for blank viii.

9. (0.5 points) Write down the number(s) for blank ix.
10. (0.5 points) Write down the number(s) for blank x.

11. (2 points) Based off the guide, how many I/Os does the entire External Merge Sort on this table take?

Now suppose your friends also want to hash the table and thus require a walk-through of the hashing process as well. Still assuming N = 100 and B = 5, fill out the following pass-by-pass guide which details the number and size of partitions created after each partitioning pass in External Hashing, assuming uniform hash functions are used in each pass.

In the first blank for each pass, write the **total** number of partitions created after the pass. In the second blank, write the number of pages in each of those partitions. If not all partition passes are needed, write 0 for all blanks in unneeded passes.

- Partitioning pass 1: ____i____ partition(s) of ____ii____ page(s) are created.
- Partitioning pass 2: ____iii____ partition(s) of ____iv____ page(s) are created.
- Partitioning pass 3: ____v____ partition(s) of ____vi____ page(s) are created.
- Partitioning pass 4: ____vii____ partition(s) of ____viii____ page(s) are created.
- Partitioning pass 5: ____ix____ partition(s) of ____x____ page(s) are created.

12. (0.5 points) Write down the number for blank i.
13. (0.5 points) Write down the number for blank ii.

14. (0.5 points) Write down the number for blank iii.
15. (0.5 points) Write down the number for blank iv.

16. (0.5 points) Write down the number for blank v.
17. (0.5 points) Write down the number for blank vi.

18. (0.5 points) Write down the number for blank vii.
19. (0.5 points) Write down the number for blank viii.

20. (0.5 points) Write down the number for blank ix.
21. (0.5 points) Write down the number for blank x.

22. (4 points) Based off the guide, how many I/Os does the entire External Hashing process on this table take? Remember that External Hashing consists of both the divide phase and the conquer phase.

7 Scratch Work (0 points)