Midterm 1: CS186, Spring 2012

Prof. J. Hellerstein

You should receive a double-sided answer sheet and a 7-page exam. Mark your name and login on ***both*** *sides of the answer sheet. For each question, place* ***only your final answer*** *on the answer sheet—do not show work or formulas there. You may remove the staple from the exam and use the backs of the questions for scratch paper.*

I. Hashing [10 points]

Imagine you are taking a CS186 test and suddenly—boom!—you become a farmer. As a farmer, you have a lot of things, so you decide to create a database to store information about all your things.

For this question, we'll examine your relation for **Animals**, where each tuple represents one animal you own, and includes a bunch of information about that animal (e.g., its name, type, weight, favorite brand of ice cream, etc.). For instance, you might have five cows, which would be represented as five individual rows with various attributes, though all would have the type "cow". Because you store so much information about each animal, you can store only **4 tuples per page.**

You want to group the animals by their type. Since you don't care about any form of ordering, you decide to **hash** the animals into groups. You have **101 buffer pages** available and there are **32,000 animals**.

1. How many times do we have run the **Partitioning** stage of hashing to hash the animals, assuming all the partitions end up being the same length? **[3pt]**
   1.
   32,000 animals means we have 8,000 pages of data. After one pass of partitioning, each partition will be 80 pages long (8000 data pages / 100 buffer pages), so they'll fit nicely into the ReHash stage.

2. How large will the average partition be as they enter the **ReHash** stage, in pages? **[3pt]**
   80.
   Calculation shown above.

Later, you need to count how many each type of animal you have, for tax reasons. You decide to use hash aggregation with hybrid hashing. For the next 2 questions, assume you have **150 buffer pages** and **40,000 animals**.

3. In order to do hybrid hashing, we need to reserve **K** pages for the in-memory hash table. We'll choose **K** by asking the question: What is the maximum **K** such that we can hash all the animals in **2** passes, using just **B-K** buffers? In other words, we want to ensure that all the animals will be hashed in **two** passes, and any buffers leftover can be used for hybrid hashing. On the answer sheet (**not here!**), mark the circle that is by the closest value of **K**.

[1pt]

A reasonable answer is 75, as follows:

In pass one we have about (B-K) pages for output partitions. We want each to be no bigger than B or so. So N <= (B-K)*B. Plugging, with B=150 and N=10,000 (40000/4), we get B-K = 66.67, or K = 150-66.67 = 83.33.

We also accepted 50:

If we have R buffer pages, then we can sort up to R^2 pages in two passes. R=B-K, so we want to solve $(B-K)^2 = N$. Plugging in, with B=150 and N=10,000 (40,000/4), we get K=50. Note that while we could technically use all B buffers during the merge phase as mentioned above, the question states to only use B-K buffers.

4. Since we're just trying to count the various animals, we only need to store the intermediate counts in our hybrid hashing table. These are tuples of the form *<animal, count>* (e.g. <'cow', 6> to represent that we've counted six cows so far). We can store 20 of these tuples per page. Given the particular **K** you chose above, what's the greatest number of **different types of animals** could there be such that we can hash them all in one pass? **[3pt]**

   1,500

   We can store 20*K different types of animals. 20*<question 3>= 1,500.

   or

   We also accepted 1,000 if you said 50 in Q3.

   We can store 20*K different types of animals. 20*<question 3> = 1,000.

II. DogSquareBookGram: A SQL Success Story [13 points]

This is your big break! Your business partner has just come to you with a paradigm-shifting social-media-meets-collective-bargaining pitch: Groupon meets Instagram for Dog Owners. As a gifted database architect and rockstar ninja hacker, it's up to you to build the site while your partner raises Series B funding.

You start by setting up tables for your human users and for their dogs. **A user can have multiple dogs, but dogs can have only one user.**

```
CREATE TABLE users (
 userid integer,
 name text,
 age integer,
 PRIMARY KEY (userid));

CREATE TABLE dogs (
 dogid integer,
 owner integer,
 name text,
 breed text,
```

```
age integer,
PRIMARY KEY (dogid),
FOREIGN KEY (owner) REFERENCES users);
```

5. It would be awesome to have a counter on your website keeping track of how many dog breeds are signed up for your service! On the answer sheet **(not here!)** mark the squares by the letters for *all* the queries that are guaranteed to return this number. **(Zero, one, or more than one may be correct.) [2.5pt]**

   a. SELECT COUNT(*) FROM dogs;

   b. SELECT COUNT(*) FROM dogs GROUP BY name;

   c. SELECT COUNT(name) FROM dogs;

   d. SELECT COUNT(dogid) FROM dogs;

   e. SELECT COUNT(DISTINCT breed) FROM dogs;

   e
   The others don't even use the "breed" column.

6. Which query should you issue to find the user id of the user with the most dogs along with the number of dogs the user owns? The query should return only one row in the case where there are multiple users with the same number of dogs. On the answer sheet **(not here!)** mark a **single** letter for the query of your choice in the box below. **[2pt]**

   a.
```
SELECT userid, COUNT(*) as cnt
  FROM dogs, users
 WHERE userid = dogid
ORDER BY cnt DESC
   LIMIT 1;
```

   b.
```
SELECT userid, MAX(dogid) as max
  FROM dogs,users
 WHERE userid = owner
GROUP BY dogid, userid
ORDER BY max DESC
   LIMIT 1;
```

   c.
```
SELECT userid, MAX(dogid) as max
  FROM dogs,users
 WHERE userid = owner
GROUP BY userid
ORDER BY max DESC
   LIMIT 1;
```

   d.
```
SELECT userid, COUNT(*) as cnt
  FROM dogs,users
 WHERE userid = owner
GROUP BY userid
ORDER BY cnt DESC
```

```
    LIMIT 1;
```

e.
```
    SELECT userid, COUNT(*) as cnt
      FROM dogs LEFT OUTER JOIN users ON userid = dogid
GROUP BY userid;
```

d

(a) and (e) equate people with dogs. (b) and (c) try to find the maximum dogid.

7. To further improve the user experience, you decide to create another table to store photos of dogs (stored as type 'bytea'):

```
CREATE TABLE photos (
  dogid integer,
  photo bytea NOT NULL, -- constraint: cannot store NULL in this field of this table
  FOREIGN KEY (userid) REFERENCES users
  FOREIGN KEY (dogid) REFERENCES dogs
);
```

Your partner absolutely adores greyhounds, so he wants to find out which users own greyhounds and haven't yet added a photo of their dog. Assume the greyhound breed is represented as "greyhound". Which SQL query answers his query? Unlike the previous query, there is **exactly one correct answer** to this question. Given the choices (a)-(e) below, mark all correct answers on the answer sheet **(not here!) [2.5pt]**

a.
```
SELECT owner
  FROM dogs
 WHERE breed = "greyhound"
   AND owner NOT IN (SELECT dogid FROM photos);
```

b.
```
SELECT owner
  FROM dogs, photos
 WHERE dogs.breed = "greyhound"
   AND dogs.dogid != photos.dogid;
```

c.
```
SELECT owner
  FROM dogs
 WHERE breed = "greyhound"
   AND dogid NOT IN (SELECT dogid FROM photos);
```

d.
```
SELECT owner
  FROM dogs LEFT OUTER JOIN photos ON dogs.dogid = photos.dogid
 WHERE photos.photo IS NULL;
```

e.
```
SELECT owner
  FROM dogs RIGHT OUTER JOIN photos ON dogs.dogid = photos.dogid
 WHERE photos.photo IS NULL;
```

c

(a) would fail to return owners who have other dogs that do have photos. (b) would fail if there's at least 1 non-matching photo in the photos table. (d) and (e) fail because they

don't select for "greyhound"s, but (d) would've succeeded if we hadn't forgotten to put that in there. Oops!

8. How many rows are returned by R x R for a given relation R of size N? Mark a single answer on the answer sheet **(not here!)**. **[3pt]**
   a. 2N
   b. 4N
   c. N^2
   d. $2N^2$
   e. 2^N

   c
   The size of a cross product is the product of the two relations, so it's just N^2.

9. True or False: $A \bowtie_\theta B = \sigma_\theta (A \times B)$. Mark the answer sheet **(not here!). [1pt]**
   True.
   This is the precise definition of theta join given in the lecture slides. If it were a natural join, you'd have to add a projection too.

10. True or False: $(A \cup B) \cup (C - D) = (B \cup (A \cup (C - D)))$. Mark the answer sheet **(not here!). [1pt]**
    True.
    Unions are commutative and associative.

11. True or False: $(A \cap (B - C)) \cap D) = (A \cap ((C - B) \cap D))$. Mark the answer sheet (**not here!). [1pt]**
    False.
    Subtraction is not commutative.

III. Sorting [11 points]

Silicon Valley is abuzz about Big Data. Last week yet another startup company was funded: BigSort.com. Their big idea is to provide a service in which users can upload a table of numbers, and the site will store the table on disk, sort it, and store the sorted results for them.

As of last week, BigSort.com had servers that use **4KB disk blocks**, and they had **200KB of memory** available for sorting. Additionally, since the company was still developing its technology, it had not parallelized its software as yet, and could only sort one file at a time. However, the founders of BigSort were very business-oriented and already laid down their pricing plan:

> BigSort.com charges users **$1 for every I/O request** that gets performed during sorting. We **do not charge to store your data on disk**! We charge only for sorting (***including the cost of writing your sorted output***.)

12. Initially, the company did not advertise, and attracted only small users. Their first user wanted to sort a file that was 160KB big. How much did BigSort charge that user? Write your answer on the answer sheet **(not here!). [3pt]**
    $80.
    B = 200/4 = 50
    N = 160/4 = 40
    We can sort this in one pass, so the number of I/Os = 2N = $80.

13. As news of BigSort.com's awesome sorting service spread, they began to attract larger and larger consumers. Their next customer had a huge file of size 300KB. How much did they charge that user? Write your answer on the answer sheet **(not here!). [3pt]**
    $80.
    B = 200/4 = 50
    N = 160/4 = 75
    This requires two passes, so we need 4N I/Os = $300.

14. Mr. X wants the biggest bang for his buck. What is the maximum size (**in KB**) of the file he could sort for $4900? Write your answer on the answer sheet **(not here!).** *(Hint: You may want to start with some "ballpark" estimates on the number of passes, and then think about cost per pass.)* **[3pt]**
    4,900 KB.
    In one pass, we can sort 50 pages. This would cost 2N = $100. We need to go deeper.
    In two passes, we can sort 50*49=2450 pages of data. This would cost 4N = $9,800.
    So we're on the right track with 2 passes, but we need to sort less data. Specifically, 4N=$4900, so N = 1,225 pages. So how many KB? 4N = 4,900 KB.

15. Coach X was the football coach of Berkeley. He had a huge table called "Footballers" that had many attributes about his players, including touchdowns scored, personal weightlifting record, favorite flower, etc. However, he didn't care about those details. All he wanted to do was to see the top 10 players by weight. Essentially he wanted to run the query to the right.

```
SELECT name, weight
FROM Footballers
ORDER BY weight DESC
LIMIT 10;
```

Which of the following optimizations could reduce the cost of executing that query while still returning the correct answer? On the answer sheet **(not here!)** mark the squares by the letters for *all* correct answers. **(Zero, one, or more than one may be correct.) [2pt]**

- ☐ a. Use Tournament Sort instead of Quicksort.
- ☐ b. Project to (name,weight) during scan.
- ☐ c. Use Hybrid Hashing.
- ☐ d. Replace Rendezvous steps with Streaming.

a, b.

(c) wouldn't help because hashing doesn't help with ORDER BY. (d) doesn't make sense—sometimes you *need* rendezvous, e.g. in sorting for ORDER BY.

(a) is likely to help since tournament sort often produces fewer runs than quicksort and (b) is almost certain to help since we could fit more records in memory.

IV. Joins [11 points]

In the spirit of Valentine's Day you have realized that dogs need love too, and have created a matchmaking service for dogs called Puppy Love Inc. You have compiled records of Male and Female dogs and stored them in two separate tables. Each record includes a "personality score" and you want to find the male-female pairs with matching "personality scores". Naturally you need to find a join algorithm! **Each record** in the table is **1KB**. The **Female table has 8,000 records** and the **Male table has 12,000 records**. The computer you are using has **816KB of memory** available and pages/buffers of size **8KB**.

16. Which table should you use as the outer relation in **simple nested loop join**? **[1pt]**
    a. Male
    b. Female

    b
    We want to use the smaller relation as the outer.

17. Using **page nested loop** join with Male as the outer relation, what is the total number of I/O's (ignoring the cost of writing the final join answer out to disk)? **[3pt]**

    1,501,500.
    [R] = (8000 records * 1KB/record) / (8 KB/page) = 1,000 pages
    [S] = (12000 records * 1KB/record) / (8 KB/page) = 1,500 pages
    [S][R] + [S] = 1500*1000 + 1500 = 1,501,500.

Due to your I/O minimizing prowess, Valentines Day is a huge success! But suddenly lots of female dogs realize they are missing out on an opportunity and swarm to your office, submitting their information. You add them to the database and the **Female table now holds 16,000 records**!

18. Which of the following join strategies requires the least I/Os for this scenario? **[1 pt]**
    a. page nested loop join with Male as the outer-loop relation
    b. page nested loop join with Female as the outer-loop relation
    c. block (chunk) nested loop join with Male as the outer-loop relation
    d. block (chunk) nested loop join with Female as the outer-loop relation

    c
    We want to use the smaller relation as the outer, and block is strictly cooler.

19. What is the total number of I/O's taken by your chosen strategy in Part 4? Ignore the cost of writing the final join out to disk. **[3pt]**
    31,500.
    [R] = (12000 records * 1KB/record) / (8 KB/page) = 1,500 pages
    [S] = (16000 records * 1KB/record) / (8 KB/page) = 2,000 pages
    [R][S]/100 + [R] = 2000*1500/100 + 1500 = 31,500.

Throughout your experience at Puppy Love you have learned some basic facts about what joins to consider. These facts are intended for non-trivial settings: i.e. B>4 pages worth of buffer space, and relations R and S of size > B.

20. Block nested loop join is _____ better than page-oriented nested loop join. **[1pt]**
    always
    Block nested loop join is page-oriented on steroids, and steroids are always good!

21. Sort-merge join is _____ better than hash-join. **[1pt]**
    sometimes
    Hash join is cooler if one relation is really small and one is very large. Sort dominates if you have lots of duplicate join keys.

22. Hybrid Hash-Join is _____ better than block-nested loops join. **[1pt]**
    sometimes
    Nested loops works for non-equijoins and hash does not. For equijoins, hash is often better since it only makes a small number of passes over each relation, whereas block nested-loops still may visit the inner relation many times. If one relation fits in memory, the two algorithms are about equivalent.

V. Streaming [5 points]

Recall Twitter's API as used by TweeQL. The calling program makes a single request, and Twitter returns a continuous, unending stream of tweets. A tweet is a 140-character message, along with a username indicating who sent the tweet, and a timestamp indicating when the tweet was sent.

Assume you make a call to the Twitter streaming API, and get back a stream of all tweets that were sent with the hash tag "#Berkeley". Imagine you want to filter these tweets.

For example, you might have the following schema for tweets from the Twitter API:

(time DATETIME, screen_name CHAR(32), text CHAR(140), hashtags CHAR(140))

Now assume you have the following schema for a stored table of "TAs":

(TA_name CHAR(128), course CHAR(16), influence FLOAT)

23. Assume you want to achieve a streaming "join" of the stored table of TAs and the continuous stream of tweets, producing a continuous stream of tuples (time, user, tweet, influence) for all tweets by Berkeley TAs with the hashtag "#Berkeley." Further assume that the TAs table fits in memory. Which join algorithms could you use? Mark the answer sheet **(not here!): (Zero, one, or more than one may be correct.) [2pt]**

- ☐ a. Sort-merge Join
- ☐ b. Simple Nested Loops Join (tweets as the "outer" relation)
- ☐ c. Block ("chunk") Nested Loops Join (TAs as the "outer" relation)
- ☐ d. Hash Join

b

(b) works: for each tweet, we scan the small set of TAs. This produces a continuous stream of outputs. We cannot use (a) or (d) because we cannot fully sort or partition an infinite tweet stream. We'd get stuck in the first phase of those algorithms forever. We cannot use (c) for a similar reason: we'd never finish the first "inner" loop over tweets and visit the second chunk of TAs.

24. Twitter doesn't guarantee that they will give you tweets back in timestamp order. Assume that you want to fix this issue by sorting the stream of tweets. Assume the stream of tweets returned by Twitter is continuous and can become arbitrarily large.

    True or False: One can devise a sorting algorithm that will output a sorted stream of Tweets. Mark your answer on the answer sheet **(not here!). [1pt]**

    False.

    There's no way to sort an infinite stream! There could be an arbitrary number of tweets between one tweet and the one that came after it in chronological order, and any solution you could cook up would have to wait *indefinitely* for the next tweet in order.

25. Now assume that Twitter guarantees that **a Tweet will be no more than k Tweets out of order**. In other words, if a Tweet is supposed to be located at position *i* in the stream, it will be in position $i \pm k$ in the worst case. Assuming **two tweets fit on a page of memory**, how many buffer pages do you need to output a sorted stream of tweets? **[2pt]**

    a. *k* buffer pages of memory
    b. *k*+3 buffer pages of memory
    c. *k*-2 buffer pages of memory
    d. 2*k*+2 buffer pages of memory
    e. It can't be done with any finite number of buffer pages

    b

    We need to store up to 2k+1 tweets (between i-k and i+k, including i), which requires k+1 buffers. We then need 2 more buffers, one for input and one for output.