COSC 460 Databases
Midterm Exam 1 (Practice Version)
Fall 2018

This is a practice exam. It is similar to the real exam in the following ways:

- 5 questions in total
- question format is similar: multiple choice, short answer, followed a SQL question and then a relational algebra question.

It is **different** from the real exam in several respects:

- It has more parts per question than you will see on the real exam. The more practice, the better!
- There's no space to write your answers... the real exam will have space.
- I didn't assign points to questions (since each question has extra parts, the point totals wouldn't be accurate anyway).

In terms of length a rough guideline for the real exam is that questions 1-4 will have 5, 2-3, 3, and 2 parts respectively.

Question	Points	Score
1	0	
2	0	
3	0	
4	0	
5	0	
Total:	0	

1. Circle your answer.

   Note: for the practice exam, it seems like all of the multiple choice questions I could think up were around algebra and SQL. On the real exam, this question will be heavily biased towards other topics. The study guide and short answer questions (Question 2) give some indication of topics that I am interested in covering.

   (a) Given relations $R(A, B)$ and $S(A, C)$, the natural join $R \bowtie S$ is equal to $R \cap S$.

   A. True **B. False**

   (b) It is possible to express the relational algebra operator intersection ($\cap$) using the join operator ($\bowtie$).

   A. True B. False

   (c) Given relations $R(A, B, C)$ and $S(D, E)$, the natural join $R \bowtie S$ is equal to $R \times S$.

   A. True B. False

   (d) It is possible to express the relational algebra operator intersection ($\cap$) using the difference operator ($-$).

   A. True B. False

   (e) The following two relational algebra expressions are equivalent (that is, they return the same answer when evaluated on any instance of the database).

$$\pi_{title,year,name}\left(\sigma_{year \geq 2001}\left(\text{Movie}\right) \bowtie \left(\text{Performs} \bowtie \text{Actor}\right)\right)$$

$$\pi_{title,year,name}\left(\sigma_{year \geq 2001}\left(\text{Movie} \bowtie \left(\text{Actor} \bowtie \text{Performs}\right)\right)\right)$$

   A. True B. False

   (f) The following two relational algebra expressions are equivalent (that is, they return the same answer when evaluated on any instance of the database).

$$\pi_{title,year}(\sigma_{length \geq 100}(Movie) \cap \sigma_{studio='Fox'}(Movie))$$

$$\pi_{title,year}(\sigma_{length \geq 100 \wedge studio='Fox'}(Movie))$$

   A. True B. False

   (g) The following two relational algebra expressions are equivalent (that is, they return the same answer when evaluated on any instance of the database).

$$\pi_{title}(\sigma_{length \geq 100}(Movie)) \cap \pi_{title}(\sigma_{studio='Fox'}(Movie))$$

$$\pi_{title}(\sigma_{length \geq 100 \wedge studio='Fox'}(Movie))$$

A. True **B. False**

(h) The following two relational algebra expressions are equivalent (that is, they return the same answer when evaluated on any instance of the database).

$$\sigma_{age>50 \vee salary<100k}(Emp)$$

$$\sigma_{age>50}(\sigma_{salary<100k}(Emp))$$

A. True **B. False**

(i) Given the relation Worker(eid, name, dept, gender), where eid is a primary key, the following two queries are equivalent (that is, they return the same answer when evaluated on any instance of the Worker relation).

```
SELECT dept
FROM Worker
WHERE gender = 'F'
INTERSECT
SELECT dept
FROM Worker
WHERE gender = 'M'

SELECT distinct x.dept
FROM Worker x, Worker y
WHERE x.gender = 'F' AND and y.gender = 'M'
```

A. True **B. False**

(j) The leaves of a B+tree are arranged sequentially on disk.

A. True **B. False**

(k) Suppose you have a unclustered B+tree index on the Emp table with composite search key $\langle age, dno \rangle$. Consider the following query

```
SELECT   *
FROM     Emp E
WHERE    E.age=30
```

It would be more efficient to answer this query using the index than simply doing a file scan. **A. True** B. False

(l) Suppose you have a clustered B+tree index on the Emp table with composite search key $\langle age, dno \rangle$. Consider the following query

```
SELECT   *
FROM     Emp E
WHERE    E.dno=5
```

It would be more efficient to answer this query using the index than simply doing a file scan. A. True **B. False**

(m) To support range queries on two different search keys, the recommended practice is to construct a clustered index (data entry alternative 1) for each key.

A. True **B. False**

(n) Circle the query descriptions below for which using a *clustered* B+tree index instead of an *unclustered* B+tree index would noticeably improve performance.

A. A range query over a reasonably large range (e.g. $3.6 < Student.gpa < 4.0$**)**

B. An equality selection on a key field (e.g. $Student.sid = 1001$)

C. An equality selection on a field with many duplicate values (e.g. $Apply.major =' Econ'$**)**

Explain your answer.

Solution: With unclustered indexes, each matching record likely requires a page read. Choices A and C have many matches whereas choice B has at most one.

2. For each question, provide a short (2-3 sentences) answer.

   (a) Why are sequential page reads faster than random page reads?

   Solution: Seek and rotational delays are minimized. See early sections of Ch. 9 of Cow book.

   (b) Why do buffer frames in the buffer pool have a pin count instead of simply a pin flag?

   Solution: See 9.4 of Cow book.

   (c) Assume you have a buffer pool with five buffer frames. Give an example set of calls to `pin(pid)` and `unpin(pid)` such that the final state of the buffer pool is different for LRU versus CLOCK. You can assume that your database has 20 pages with page ids $1, 2, \ldots, 20$. An example (incorrect) sequence is: `pin(1), pin(2), pin(3), pin(4), pin(5), unpin(1), pin(6)`.

   Solution: `pin(1), pin(2), pin(3), pin(4), pin(5), unpin(5), unpin(4), pin(6)`

   Clock will evict 4 even though 5 is the LRU.

   (d) The goal of a record id is to uniquely identify a record. A simple record id format is the pair (pid, offset), where pid is the id of the page in which the record is stored, and offset is the byte offset of the record within that page. However, instead of using the above simple record id format, most systems use the more complex (pid, slot number) format for record ids. Why? Your answer should include two different reasons that together justify the (pid, slot number) format.

   Solution: p. 329 of Cow book.

   (e) What is *prefetching* and why is it important?

   Solution: See p. 322 of Cow book.

   (f) Suppose a page is storing variable-length records and a record r is *modified* causing its size to increase. Assume that the byte immediately following the bytes allocated to r is occupied by some other record s. Describe how the data storage is rearranged to make room for the modified record under two different scenarios. Scenario 1: the page has free space; Scenario 2: the page has no free space.

Solution: Scenario 1: see p. 329 of Cow book; scenario 2: see p. 332.

(g) What is "logical data independence" and why is it important?

Solution: See 1.5.3 of Cow book.

(h) Why does the database system usually implement its own buffer manager (instead of relying on the operating system's file system cache)? Give two different reasons.

Solution: See 9.4.2 of Cow book. Reasons include the desire to prefetch and the need to force.

(i) Given two relations $R1$ and $R2$, where $R1$ contains $N1$ tuples, $R2$ contains $N2$ tuples, and $N2 > N1 > 0$, give the minimum and maximum possible sizes (in tuples) for the resulting relation produced by each of the following relational algebra expressions. In each case, state any assumptions about the schemas for $R1$ and $R2$ needed to make the expression meaningful:

i. $R1 \cup R2$

- Min size:
- Max size:
- Schema assumptions:

Solution:

- Min size: $N2$
- Max size: $N1 + N2$
- Schema assumptions: matching schema

ii. $R1 - R2$

- Min size:
- Max size:
- Schema assumptions:

Solution:

- Min size: 0
- Max size: $N1$

- Schema assumptions: matching schema

iii. $R1 \bowtie R2$

- Min size:
- Max size:
- Schema assumptions:

Solution:

- Min size: 0
- Max size: $N1 \times N2$
- Schema assumptions: none

iv. $R1 \times R2$

- Min size:
- Max size:
- Schema assumptions:

Solution:

- Min size: $N2 \times N1$
- Max size: $N2 \times N1$
- Schema assumptions: none

v. $\sigma_{a=5}(R1)$

- Min size:
- Max size:
- Schema assumptions:

Solution:

- Min size: 0
- Max size: $N1$
- Schema assumptions: a is attribute of $R1$

vi. $\pi_a(R1)$

- Min size:
- Max size:
- Schema assumptions:

Solution:

- Min size: 1
- Max size: $N1$
- Schema assumptions: a is attribute of $R1$

(j) Consider following two relational algebra expressions over relation $R(A, B)$

$$\pi_A(\sigma_{A=5}(R \bowtie \rho_{B\to C}(R)))$$

$$\sigma_{A=5}(\pi_A(R))$$

If they are are equivalent (that is, they return the same answer when evaluated on any instance of the database), then write "equivalent." Otherwise, write "not equivalent" and construct a small instance on which the above expressions return different answers.

Solution: Equivalent.

(k) Consider following two relational algebra expressions over relation $R(A, B)$

$$\pi_A(\sigma_{A=5\wedge B\neq C}(R \bowtie \rho_{B\to C}(R)))$$

$$\sigma_{A=5}(\pi_A(R))$$

If they are are equivalent (that is, they return the same answer when evaluated on any instance of the database), then write "equivalent." Otherwise, write "not equivalent" and construct a small instance on which the above expressions return different answers.

Solution: Not equivalent. If R is a single tuple (5, x) where x is anything, the first returns empty set whereas the second returns (5).

3. SQL

Consider the following relational schema about a social network centered around college students and alumni. People can be friends, post messages, and like each other's messages.

```
Users(uid, name, school, year)
Friends(uid1, uid2)
Posts(pid, uid, text, datetime)
Likes(uid, pid, datetime)
```

If two users with ids 123 and 456 are friends, then (123, 456) and (456, 123) will both appear in the Friends relation. Please use U, F, P, and L for the relation names in your queries.

For each of the following, write a **SQL query** that produces the desired information.

(a) Pairs of names such that the two people are friends and Colgate students (be sure to avoid duplicates).

Solution:

```
select U1.name, U2.name from
U as U1, F, U as U2
where U1.uid = F.uid1, F.uid2 = U2.uid and
U1.uid < U2.uid and
U1.school = "Colgate" and U2.school = "Colgate"
```

(b) Find posts that are authored by a Colgate student but liked by a Bucknell student.

Solution:

```
SELECT pid
FROM Posts P, Likes L, Users U1, Users U2
WHERE U1.school = 'Colgate' AND U2.school = 'Bucknell' AND
U1.uid = P.uid AND U2.uid = L.uid AND
L.pid = P.pid
```

(c) Find schools with at least two students. Avoid duplicates.

Solution:

```
SELECT DISTINCT school
FROM Users U1, Users U2
WHERE U1.school = U2.school AND
U1.uid < U2.uid
```

(d) Find user ids of users that have no friends.

Solution:

```
SELECT uid FROM Users
EXCEPT
SELECT uid1 FROM Friends
```

(e) Find ids of posts that have been liked by a current student and an alumnus. Name the return column TimelessPosts

Solution:

```
SELECT pid AS TimelessPosts
FROM Users U, Likes L WHERE U.uid = L.uid and U.year < 2019
INTERSECT
SELECT pid
FROM Users U, Likes L WHERE U.uid = L.uid AND
U.year <= 2022 AND U.year >= 2019
```

4. Algebra

   Use the same schema as described in Question 3, repeated here for reference:

```
Users(uid, name, school, year)
Friends(uid1, uid2)
Posts(pid, uid, text, datetime)
Likes(uid, pid, datetime)
```

   For each of the following, write a **relational algebra expression** that computes the desired information.

   (a) Find "Colgate groupies": user ids of those users who are friends with *everyone* who went to Colgate.

   Solution:

$$
\begin{aligned}
&uids \leftarrow \pi_{uid}(U) \\
&gaters \leftarrow \pi_{uid}(\sigma_{school='Colgate'}(U)) \\
&pairs \leftarrow \rho_{uid1,uid2}(uids \times gaters) \\
&missing \leftarrow \pi_{uid1}(pairs - F) \\
&all \leftarrow uids - missing
\end{aligned}
$$

   (b) Find "Colgate stalkers": people who like the posts of one or more Colgate students but are not friends with them (i.e., not friends with the authors of the posts they like).

   Solution:

$$
\begin{aligned}
&ColgatePosts \leftarrow \pi_{pid}(\sigma_{school=Colgate}(U \bowtie P)) \\
&ColgateLikers \leftarrow \pi_{uid}(L \bowtie ColgatePosts) \\
&ColgateFriends \leftarrow \pi_{uid2}(\sigma_{school=Colgate}(U \bowtie_{uid=uid1} F)) \\
&ColgateLikers \ - \ ColgateFriends
\end{aligned}
$$

   (c) Find "Colgate polyannas": people who like every post written by a Colgate student.

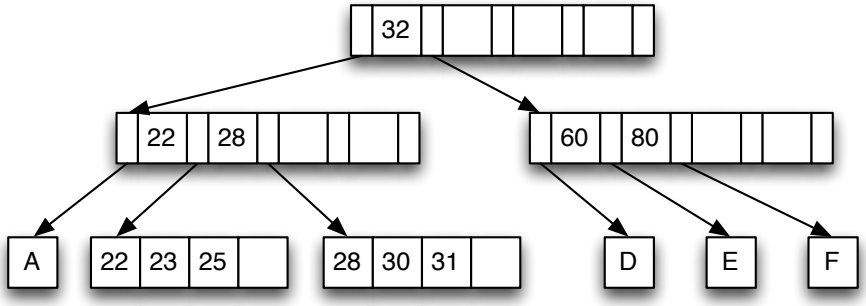
   Solution:

$$
\begin{aligned}
&ColgatePosts \leftarrow \pi_{pid}(\sigma_{school=Colgate}(U \bowtie P)) \\
&pairs \leftarrow \pi_{uid}(U) \times ColgatePosts \\
&missing \leftarrow \pi_{uid}(pairs \ - \ \pi_{uid,pid}(L)) \\
&polyannas \leftarrow \pi_{uid}(U) - missing
\end{aligned}
$$

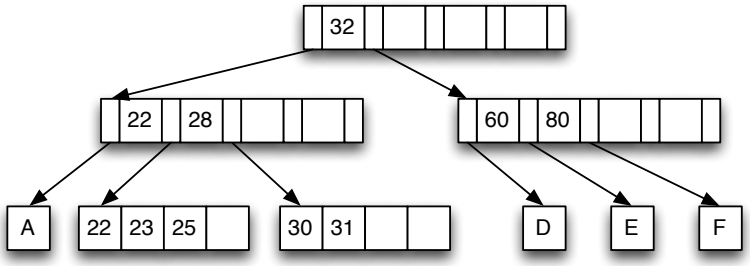
5. B+Trees

(a) Consider the B+tree shown below. The tree has order = 2. Each node of the tree is labeled: the root is R and the leaves are A, B, C, D, E. When asked to draw a tree, you can simply draw a node representing the label (like A, D, E and F) if the internal details of that node are unchanged from this figure.

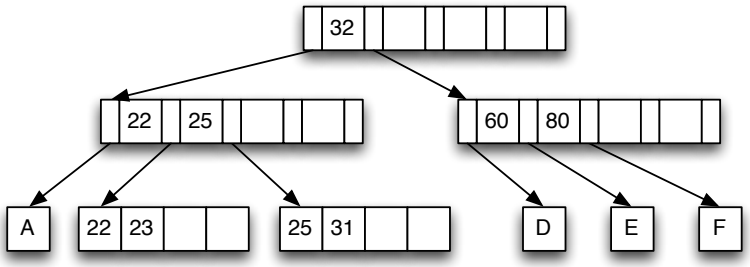
Show the trees that will result from deleting the following entries: 28, 30, and 25.



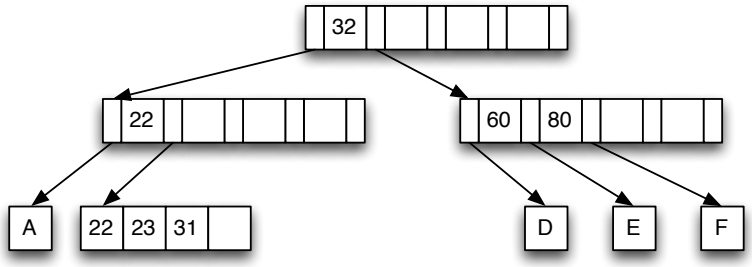
Solution: After deleting 28: simply delete 28



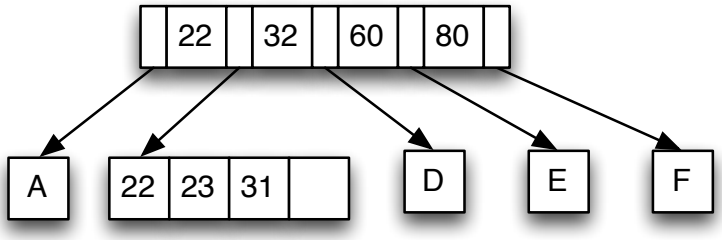
After deleting 30: do redistribution



During delete of 25: merge children



After delete of 25: merge internal nodes, drop root



(b) The height of a B+-tree depends on a number of factors. Suppose you know the number of records in the relation you want to index and the size (in bytes) of the attribute(s) you want to index on. What other factors determine the height? Give a formula for the height of the tree. Assume data entry alternative 2.

Solution:

- s size of search key (in bytes)
- p size of tree node pointer (page id, typically a table id and page number, 8 bytes)
- I size of index entry $(s + p)$
- d size of data entry alternative 2 (search key of size s + recordid which is typically a page id plus slot number, so s + 12 bytes)
- P size of page (in bytes)
- c fill level (fraction of how full each page... as book discussed a real-world B+tree is typically around 2/3 full.)
- n size of input (number of records)

First, figure out number of leaf pages B. This depends on all of the above factors.

- leaf max capacity: x data entries per page where $x = P/d$
- assume fill level c, then $x \leftarrow xc$
- B is number of leaf pages: $B = \lceil n/x \rceil$

Second, figure out fan in F.

- node max capacity: F index entries per page where $F = P/(s + p)$
- assume fill level c, then $F \leftarrow Fc$

Third, the formula for height is $\lceil \log_F B \rceil$.

(c) Is a B+tree a balanced tree? Explain why or why not.

Solution: Yes. All leaves are guaranteed to be at the same depth.