CS 186/286 Spring 2018 Midterm 1

- Do not turn this page until instructed to start the exam.
- You should receive 1 single-sided *answer sheet* and a 13-page *exam packet*.
- All answers should be written on the answer sheet. The exam packet will be collected but not graded.
- You have *80 minutes* to complete the midterm.
- The midterm has *4 questions*, each with multiple parts.
- The midterm is worth a total of *75 points*.
- 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, **do not mark the box with an X or checkmark**.
- Use the blank spaces in your exam for scratch paper.
- You are allowed **one** 8.5" $\times$ 11" double-sided page of notes.
- No electronic devices are allowed.

1 SQL/Relational Algebra (28 points)

For the following questions, we consider the following schema:

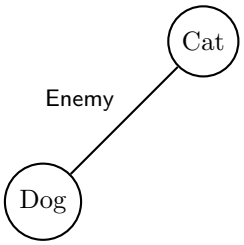
```
CREATE TABLE Player(
  pid INTEGER PRIMARY KEY,
  name TEXT
);

CREATE TABLE Relationship(
  pid1 INTEGER REFERENCES Player(pid),
  pid2 INTEGER REFERENCES Player(pid),
  type TEXT,
  time TIMESTAMP,
  PRIMARY KEY(pid1, pid2, time)
);
```

Players may be part of a relationship (with the "type" being either "Friend" or "Enemy") with other players.

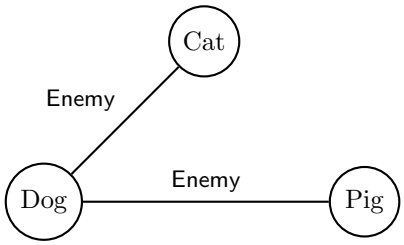
A single row in the `Relationship` table represents the start of a relationship between two players. The most current relationship between players A and B is the row in Relationship with A, B, and the latest timestamp.

Consider the following relationships:

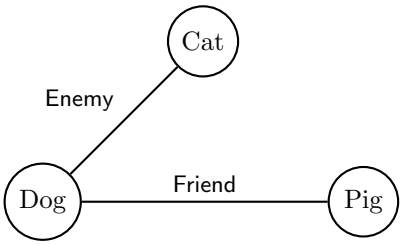
At `2018-03-01 11:40:00`:



At `2018-03-01 11:50:00`:



At `2018-03-01 12:00:00`:



The corresponding tables are:

Player

pid	name
1	Cat
2	Dog
3	Pig

Relationship

pid1	pid2	type	time
1	2	Enemy	`2018-03-01 11:40:00`
2	1	Enemy	`2018-03-01 11:40:00`
2	3	Enemy	`2018-03-01 11:50:00`
3	2	Enemy	`2018-03-01 11:50:00`
2	3	Friend	`2018-03-01 12:00:00`
3	2	Friend	`2018-03-01 12:00:00`

Relationships are always mutual: if there is a row with `pid1=3` and `pid2=4`, there will be a corresponding row with the same `type`/`time` and with `pid1=4` and `pid2=3`.

Assume that a player is never in a relationship with themself (Dog cannot be friends with Dog, and Cat cannot be enemies with Cat).

1. (4 points) We wish to find players that have started enemy relationships (with the same or with different players) at least 50 times. This could be between the same pair of players at least 50 times, or between at least 50 others, etc.

   Mark all of the following queries that do this.

   A.
   ```
   SELECT Player.*
     FROM Player
   INNER JOIN Relationship ON pid1 = pid
     WHERE type = 'Enemy' AND COUNT(*) >= 50
   GROUP BY pid, name;
   ```

   B.
   ```
   SELECT Player.*
     FROM Player, Relationship
    WHERE pid = pid1 AND type = 'Enemy'
   GROUP BY pid, name
     HAVING COUNT(*) >= 50;
   ```

   C.
   ```
   SELECT Player.*
     FROM Player
   INNER JOIN Relationship ON pid1 = pid
     WHERE COUNT(*) >= 50;
   GROUP BY pid, name, type
     HAVING type = 'Enemy';
   ```

   D.
   ```
   SELECT Player.*
     FROM Player, Relationship
   GROUP BY pid, pid1, type, name
     HAVING pid = pid1 AND type = 'Enemy' AND COUNT(*) >= 50;
   ```

2. (6 points) Now instead, we decide that we want to fetch the number of times each player has started an enemy relationship (if a player became enemies with the same player twice, count it twice). If a player has never started an enemy relationship, the count should be 0.

   Mark all of the following queries that do this.

   A.
   ```
   SELECT Player.*, COUNT(*)
     FROM Player
   LEFT JOIN Relationship ON pid1 = pid
     WHERE type = 'Enemy'
   GROUP BY pid, name;
   ```

   B.
   ```
   SELECT Player.*, COUNT(pid1)
     FROM Player
   LEFT JOIN Relationship ON pid1 = pid
   GROUP BY pid, name, type
     HAVING type = 'Enemy';
   ```

   C.
   ```
   SELECT Player.*, COUNT(pid1)
     FROM Player
   LEFT JOIN Relationship ON pid1 = pid
     WHERE type = 'Enemy' OR type IS NULL
   GROUP BY pid, name;
   ```

   D.
   ```
   SELECT Player.*, COUNT(*)
     FROM Player
   LEFT JOIN Relationship ON pid1 = pid
     WHERE type = 'Enemy' OR type IS NULL
   GROUP BY pid, name;
   ```

3. (6 points) Now, instead of counting the number of times a player has become enemies with other players, we wish to find the last player(s) with whom each player has entered into a relationship. Omit players that have never started a relationship. If the player started a relationship with multiple players at the same time, return a row for each one.

   Mark all of the following queries that do this.

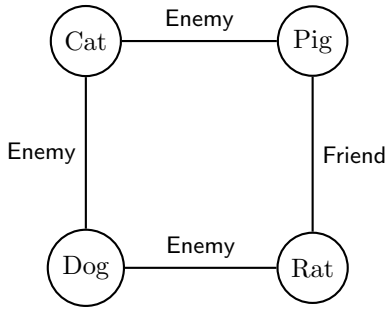
   A.
```
SELECT pid1, pid2
 FROM Relationship R1
WHERE EXISTS (
    SELECT 1 FROM Relationship R2
    WHERE R1.pid1 = R2.pid1
      AND R1.time > R2.time
);
```

   B.
```
SELECT pid1, pid2
 FROM Relationship R1
WHERE time IN (
    SELECT MAX(R2.time) FROM Relationship R2
    WHERE R1.pid1 = R2.pid1
);
```

   C.
```
SELECT pid1, pid2
 FROM Relationship R1
WHERE R1.time >= ALL (
    SELECT R2.time FROM Relationship R2
    WHERE R1.pid1 = R2.pid1
);
```

   D.
```
SELECT pid1, pid2
 FROM Relationship R1
WHERE time >= ALL (
    SELECT MAX(R2.time) FROM Relationship R2
    GROUP BY R2.pid1
);
```

4. (8 points) We now wish to find enemies of enemies at the current time. That is, we wish to find players A and C, such that the latest relationship in our table between A and some third player B is "Enemy", and the latest relationship between B and C is also "Enemy".



In the diagram above, Cat and Rat are enemies of enemies, because Cat is an enemy of Dog, and Dog is an enemy of Rat.

Mark all of the queries that fetch all current enemy of enemy pairs.

A.
```
SELECT r1.pid1, r2.pid2
  FROM Relationship r1, Relationship r2, Relationship r3, Relationship r4
 WHERE r1.pid2 = r2.pid1 AND r1.type = r2.type
   AND r1.pid1 = r3.pid1 AND r1.pid2 = r3.pid2 AND r1.type = r3.type
   AND r2.pid1 = r4.pid1 AND r2.pid2 = r4.pid2 AND r1.type = r4.type
   AND r1.pid1 <> r2.pid2 AND r1.type = 'Enemy'
GROUP BY r1.pid1, r2.pid2, r1.time, r2.time
  HAVING MAX(r3.time) <= r1.time AND MAX(r4.time) <= r2.time;
```

B.
```
SELECT r1.pid1, r2.pid2
  FROM Relationship r1, Relationship r2, Relationship r3, Relationship r4
 WHERE r1.pid1 = r3.pid1 AND r1.pid2 = r3.pid2
   AND r2.pid1 = r4.pid1 AND r2.pid2 = r4.pid2
   AND r1.pid1 <> r2.pid2 AND r1.type = 'Enemy' AND r2.type = 'Enemy'
   AND r1.pid1 <> r2.pid2 AND r1.type = 'Enemy'
GROUP BY r1.pid1, r2.pid2, r1.time, r2.time
  HAVING MAX(r3.time) <= r1.time AND MAX(r4.time) <= r2.time;
```

C.
```
SELECT r1.pid1, r2.pid2
  FROM Relationship r1, Relationship r2, Relationship r3, Relationship r4
 WHERE r1.pid2 = r2.pid1
   AND r1.pid1 = r3.pid1 AND r1.pid2 = r3.pid2
   AND r2.pid1 = r4.pid1 AND r2.pid2 = r4.pid2
   AND r1.pid1 <> r2.pid2 AND r1.type = r2.type
GROUP BY r1.pid1, r2.pid2, r1.type, r1.time, r2.time
  HAVING MAX(r3.time) <= r1.time AND MAX(r4.time) <= r2.time AND r1.type = 'Enemy';
```

D.
```
SELECT r1.pid1, r2.pid1
  FROM Relationship r1, Relationship r2, Relationship r3, Relationship r4
 WHERE r1.pid2 = r2.pid2
   AND r1.pid1 = r3.pid1 AND r1.pid2 = r3.pid2
   AND r2.pid1 = r4.pid1 AND r2.pid2 = r4.pid2
   AND r1.pid1 <> r2.pid1 AND r3.type = r4.type
GROUP BY r1.pid1, r2.pid1, r3.type, r1.time, r2.time
  HAVING MAX(r3.time) <= r1.time AND MAX(r4.time) <= r2.time AND r3.type = 'Enemy';
```

For the following problems, we use P to denote the Player table and R to denote the Relationship table. We additionally assume that for any pair of `pid`s (e.g. 1 and 2), there is at most one pair of rows in the Relationship table (with `pid1=1, pid2=2` and `pid1=2, pid2=1`). In other words, we are only storing the most current relationship.

5. (3 points) We wish to find the name of all players that are not enemies of enemies with any other player. Mark all of the following relational algebra expressions that satisfy this query.

   A. $\pi_{\text{name}}(\sigma_{\text{type}=\text{`}Enemy\text{'}}(P - \pi_{\text{pid,name}}(P \bowtie \pi_{\text{pid1,pid2,type}}(R) \bowtie \rho_{\text{pid1}\to\text{pid}}(R))))$

   B. $\pi_{\text{name}}(P - \pi_{\text{pid,name}}(\sigma_{\text{type}=\text{`}Enemy\text{'}}(P \bowtie \pi_{\text{pid1,pid2,type}}(R) \bowtie \rho_{\text{pid1}\to\text{pid}}(R))))$

   C. $\pi_{\text{name}}(P) - \pi_{\text{name}}(\sigma_{\text{type}=\text{`}Enemy\text{'}}(\pi_{\text{pid1,pid2,type}}(P \bowtie R) \bowtie \rho_{\text{pid1}\to\text{pid}}(R)))$

6. (1 point) True or false: the two relational algebra expressions are equivalent (on all possible databases).

   $\sigma_{\text{type}=\text{`Enemy'}}(R) - (\sigma_{\text{type}=\text{`Enemy'}}(R) - \sigma_{\text{time}<\text{`2018-01-01'}}(R))$

   $\sigma_{\text{time}<\text{`2018-01-01' AND type}=\text{`Enemy'}}(R)$

2 File Organization (6.5 points)

2.1 True and False

1. (2.5 points) Specify correct choice(s) by filling in the corresponding box on the answer sheet.
    A. In order to interact with all hard disks, computers use an explicit API.
    B. Hard drives need to constantly re-organize the data stored on them (wear-levelling) to prevent failure.
    C. For flash drives, read and writes are both equally fast since these drive use NAND technology instead of traditional spinning magnetic disks.
    D. Heap files are preferable to Sorted files when fast writes are required.
    E. For pages holding variable length records, the size of the slot directory never changes.

2.2 Records and Pages

Consider the following relation:

```
CREATE TABLE Universities{
    name TEXT
    abbrev CHAR(3)
    addr TEXT
    zip CHAR(5)
    phone TEXT
    enrolled INTEGER
}
```

1. (1 point) How big, in bytes is the smallest possible record? Assume the record header takes up 4 bytes and integers are 4 bytes long.
2. (3 points) Assuming each record is 22 bytes long and we have access to 5 pages each of size 4KB (1KB = 1024B) where each page has a footer containing 6 bytes of metadata (including the free space pointer) as well as a slot directory, how many records can we fit on the 5 pages? Assume integers are 4 bytes long . Assume that records cannot span multiple pages.

3 Indices (20.5 points)

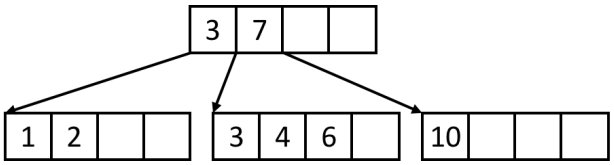
For this section, remember that **the height of a one level tree is defined to be 0, the height of a tree with two levels is defined to be 1, and so on.**

3.1 True and False

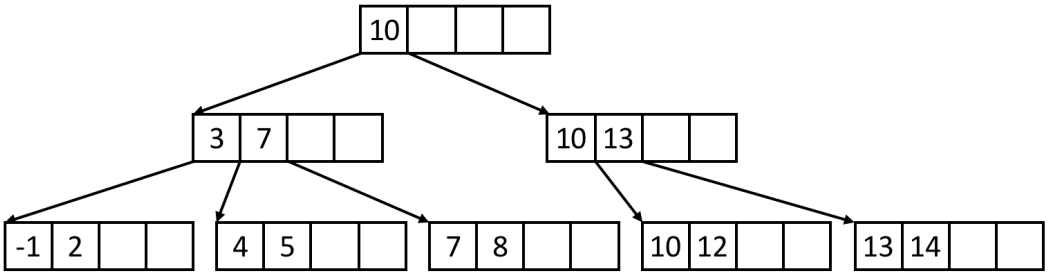
1. (2.5 points) Specify correct choice(s) by filling in the corresponding box on the answer sheet.
   A. The maximum fanout of a B+ tree is equal to 2d where d is its order.
   B. When an insertion into a B+ tree causes a node to split, the height of the tree always increases.
   C. As the number of keys in a B+ tree increases, IO cost for searches grows logarithmically with the number of keys.
   D. Clustered and unclustered B+ trees provide the same performance benefits for all queries.
   E. Bulkloading for B+ trees is efficient because we only keep the left most branch in memory for insertions and can disregard the rest of the tree.

2. (4 points) For the B+ trees below, determine which of the trees are valid B+ trees based on the invariants we learned in class, and fill in its corresponding box on the answer sheet. Assume that there were no deletes. Note, we follow the right branch when the split key equals the search key.

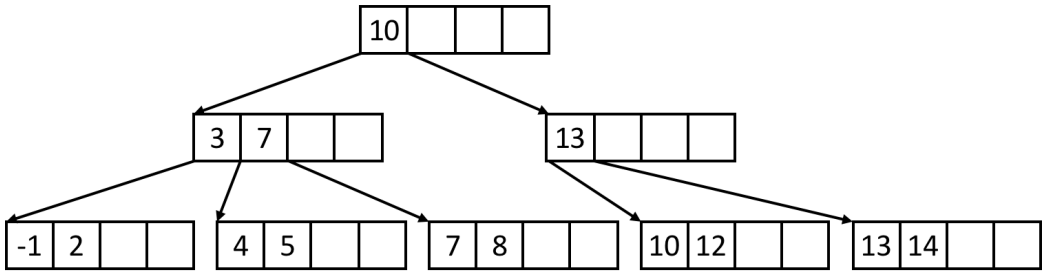
A.



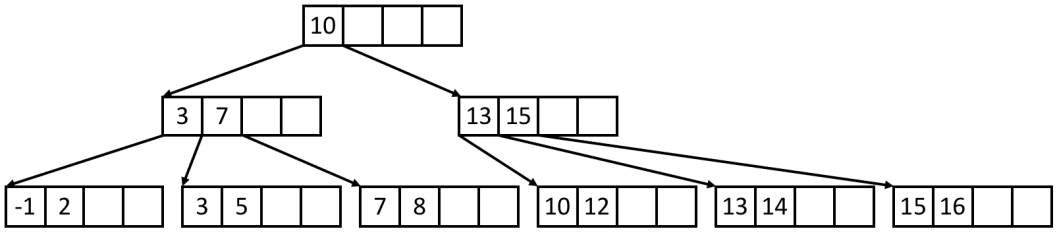
B.



C.



D.

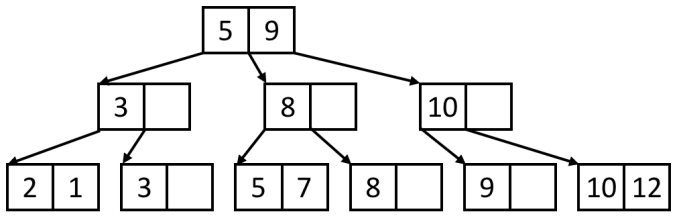


3. (3 points) Suppose we have an initially empty, order d=1 B+ tree, and the following set of keys:

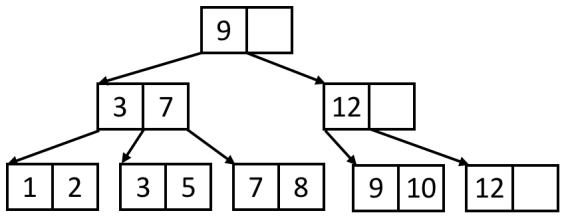
$$3, 9, 8, 7, 2, 1, 5, 10, 12$$

Which of the following are possible B+ trees after inserting all of the above elements in some order? Again, we follow the right branch when the split key equals the search key. When splitting a node with an odd number of keys, assume the new right node receives one more key than the original left node.

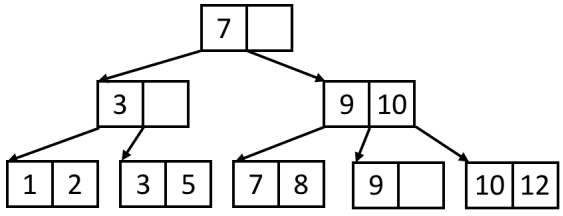
A.



B.



C.



3.2 Short Answer

1. (.5 points) What is the maximum fanout of an order x+1 B+ tree?
2. (.5 points) What is the maximum number of keys an order 2 B+ tree with height 2 can store?
3. (.5 points) For a height 3 alternative 2 B+ tree, how many IOs are required for an equality search on the index key given that the index key is a primary key?

3.3 Bulk Loading

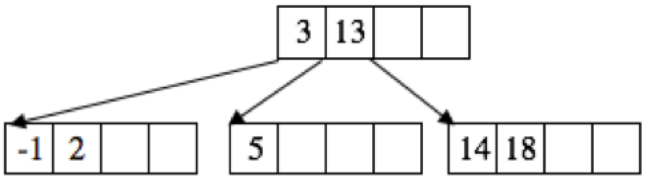
What is the resulting B+ tree after bulk loading the following keys? Assume we are building an order d=1 index with a minimum leaf node fill factor of 2/3. When splitting an inner node, perform the split the same way we did in the homework – with an internal node fill factor of 1/2.

$$3, 8, 5, 19, 20, 11, 7$$

Answer the questions below:

1. (.5 points) What is the height of the resulting B+ tree?
2. (.5 points) How many inner nodes are there in the resulting B+ tree?
3. (.5 points) How many leaf nodes are there in the resulting B+ tree?
4. (.5 points) What key(s) are in the leftmost leaf node? Separate keys with commas.
5. (.5 points) What key(s) are in the rightmost leaf node? Separate keys with commas.
6. (1 point) Write out the contents of the resulting B+ tree in a level-order traversal. Remember that a level-order tree traversal enumerates the nodes of a tree one level at a time, and from left to right on each level. That is, first the root, then the children of the root from left to right, and so on. Separate keys with commas.

Use the B+ tree below for the following questions:



7. (2 points) What is the maximum number of keys we can insert before changing the height of the tree?
8. (2 points) What is the minimum number of keys that we can insert to change the height of the tree?
9. (2 points) How many total IOs are required to insert the keys 15, 16, 17? Remember that reading in a page into memory and writing the page back to disk counts as 2 IOs (1 read and 1 write). Additionally, assume each insert starts with an empty buffer pool.

4 Buffers/Sorting and Hashing (20 points)

For any question that asks which pages are in the buffer pool at some point in time, please list the pages in **alphabetical order.**

1. (5 points) Assume we have 4 empty buffer frames and the following access pattern, in which pages are immediately unpinned:

   $A, C, D, F, D, C, B, C, A, E, C, D$

   a) Which pages are in the buffer pool at the end if we use MRU cache policy?

   b) How many cache hits will there be after MRU cache policy?

   c) Which pages are in the buffer pool at the end if we use Clock cache policy?

   d) How many cache hits will there be after Clock cache policy?

   e) How many reference bits are set after Clock cache policy?

2. (3 points) Assume we have 4 empty buffer frames and the following access pattern, in which pages are immediately unpinned:

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

   a) Which pages are in the buffer pool at the end if we use LRU cache policy?

   b) How many cache hits will occur if we use LRU cache policy?

   c) True or False: For any sequential access pattern and buffer size, LRU will always have a cache hit rate strictly less than MRU.

3. (3 points) True or False:

   a) To finish the final pass of external hashing (conquer), we use a coarse-grained hash function, h_r, that is different from the hash functions used in the previous passes.

   b) De-duplicating a file using hashing is always more efficient than sorting in terms of I/O cost.

   c) Given any hashed file, there is some permutation of pages such that the resulting file is sorted.

4. (6 points) Suppose the size of a page is 4KB, and the size of the memory buffer is 320KB. We want to perform external sorting for a dataset. Answer the following questions given these configurations.

   a) What is the maximum size of the dataset (in KB) that can be sorted in one pass?

   b) What is the maximum size of the dataset (in KB) that can be sorted in n passes for $n >= 1$? (Use n in your solution)

   c) If the size of the dataset is 800KB, how many disk I/O do we need to sort this dataset?

5. (3 points) We are given a table containing personal information of UC Berkeley students wish a size of 100,000 KB. Suppose we want to group students by their year of birth. We have 51 buffer pages available, and the size of each page is 100KB.

   a) How large will the average partition be as they enter the second phase of external hashing, in pages?

   b) Assume that our table only contains information about undergraduates at UC Berkeley, and we would like to eliminate duplicate students using hashing. Rank the following hash functions from best to worst for accomplishing this task (in your answer, list the letters of the hash functions from left to right with the leftmost letter being the best hash function and the rightmost one being the worst). We consider a hash function with less collisions (different records ending up in the same bucket) to be "better".

```
CREATE TABLE Enrolled {
    firstname TEXT,
    lastname TEXT,
    addr TEXT,
    sid INTEGER,
    gender CHAR(1),
    year_of_birth INTEGER
}
```

   A. h(record) = record.year_of_birth % 10

   B. h(record) = record.sid % 10

   C. h(record) = to_int(record.gender) % 10

   D. h(record) = to_int(record.firstname[0]) % 10