Name: _____SOLUTION_____

Midterm 1: CS186, Spring 2016

Class Login: cs186- _____ _____ _____

You should receive 1 double-sided answer sheet and an 10-page exam. Mark your name and login on ***both sides of the answer sheet****,* ***and in the blanks above****. For each question, place* ***only your final answer*** *on the answer sheet—do not show work or formulas there. You may use the blank spaces for scratch paper, but* ***do not tear off any pages****. You will* ***turn in both question and answer sheets****.*

I. Storage: Disk, Files, Buffers [11 points]

1. [3 points] Write down the letters of true statements *in alphabetical order.* (If none are true, write ∅.)
   A. When querying for a 16 byte record, exactly 16 bytes of data is read from disk. False, a page's worth of data is always read from disk.
   B. Writing to an SSD drive is more costly than reading from an SSD drive. True, writes to an SSD can involve reorganization to avoid uneven wear and tear.
   C. In a heap file, all pages must be filled to capacity except the last page. False, there is no such requirement. With clustered indices, it is common to keep heap file pages 2/3 full to accommodate inserts.
   D. If the file size is smaller than the number of buffer frames, a sequential scan of the file using either MRU or LRU (starting with an empty buffer pool) will have the same hit rate. True, our eviction policy doesn't matter because the file fits in memory.
   E. Assuming integers take 4 bytes and pointers take 4 bytes, a slot directory that is 256 bytes can address 64 records in a page. False, as shown in lecture, an entry in slot directory is 8 bytes, because a single entry consists of both a pointer and an integer (length).
   F. In a page containing fixed-length records with no nullable fields, the size of the bitmap never changes. True, the size of the records is fixed, so the number we can fit on a page is fixed.

2. [2 points] Write down the true benefits of using a record header for **variable** length records *in alphabetical order* (or if none are true benefits, write ∅.)
   A. Does not need delimiter character to separate fields in the records. True.
   B. Always matches or beats space cost when compared to fixed-length record format. False.
   C. Can access any field without scanning the entire record. True.

D. Has compact representation of null values. True.

3. [6 points] Assume we have 4 empty buffer frames and the following access pattern, in which pages are immediately unpinned.

T A M E T E A M M A T E M E A T L I D

Use the replacement policy listed, and list the four pages in the buffer pool at the end, *in alphabetical order.* Hint: you don't need to draw a big chart for every access — look for patterns.

A. MRU. ADEM
B. LRU. DILT
C. Clock. (*Assume the clock hand starts on the first buffer and does not move unless a page needs to be replaced.).* DEIL

This space left intentionally blank for scratch work.

II. Joins [12 points]

1. [4 points] Alphabetically, write down the letters of statements that apply (or write ∅.)
   A. Sometimes, adding more memory to our system **will <u>not</u>** reduce I/O costs for a sort-merge join. True. A simple counter example is joining two 1-tuple relations.
   B. A Grace hash join will always perform better than a naïve hash join. False. Consider the same example from (A).
   C. Sometimes, replacing an Alternative 2 index with an Alternative 1 index on the same key will speed up an index-nested-loops join. True. Alternative 1 lookups don't require the additional IOs to follow the record id (rid) to the data pages, which is necessary with an Alternative 2 index.
   D. A Grace hash join can often complete in 2 passes if the size of the smaller relation is less than roughly the square of the number of buffers available for the join. True, if we assume a hash function which spreads evenly.

For the following questions in this section (Joins), assume that we are streaming our join output to a terminal. **Consider the cost of the initial table scan, but *do not consider the cost of writing the final output***.

We have the following schema:

```
CREATE TABLE Cheesemakers (
cm_id INTEGER PRIMARY KEY,
name VARCHAR(33),
ranking INTEGER,
skill_level INTEGER)
```

```
CREATE TABLE Products (
cheese_id INTEGER PRIMARY KEY,
cm_id INTEGER REFERENCES Cheesemakers,
ctype VARCHAR(16),
smelliness INTEGER,
cheesiness INTEGER)
```

Until instructed otherwise, assume that:

- Cheesemakers has **[C] = 500 pages**
- Products has **[P] = 2000 pages**
- we fix **B = 102 pages** of memory for computing joins.

Consider the following query:

```
SELECT C.name, C.ranking, P.ctype, P.smelliness, P.cheesiness
FROM Cheesemakers C, Products P
WHERE C.cm_id = P.cm_id
```

2. [4 points] Using the smaller relation as the “outer” one, what is the I/O cost of using a block nested loops join to evaluate the query above? Please provide the final number.

500 + 5 * 2000 = 10500 I/Os

3. [4 points] What is the I/O cost of using a sort-merge join to evaluate the query above? Please provide the final number. (Remember to take advantage of the “important refinement” discussed in lecture for merge-joining partitions during the last pass of sort!)

Read, write, read/merge: 3 * (500 + 2000) = 7500 I/Os

This space left intentionally blank for scratch work.

III. Sort/Hash [16 points]

For this question, consider our table of Products from the previous Joins section, but assume:

- **[P] = 2000 pages**
- **p_C = 100 tuples/page**
- **B = 40 pages** of buffer
- In all parts, we'll use **QuickSort** for our internal sort algorithm.
- **Include the cost of the initial scan *and* cost of writing output in your I/O calculations.**

1. [3 points] Alphabetically, write down the letters of statements that apply (or write ∅.)
   A. Given a buffer of size B, the largest file you can sort in a single pass is B. True. This is an internal sort.
   B. All files can be externally sorted, whereas not all files can be externally hashed. True. Consider a file with very many duplicates, such that we cannot fit all duplicates in a partition.
   C. For sort-merge joins, quicksort is always a better choice than heapsort for the internal sort algorithm. False. Recall that a tournament sort will produce runs twice as long - half as many runs, which could result in fewer passes.

2. [6 points] First, let's sort our table of Products (P) using the external algorithm we learned in lecture.
   A. How many passes are needed to sort this file? 3. ceil(log_39(ceil(2000/40)))+1 = 3. We accepted answers written as an expression, only if the ceilings were correct (i.e. your expression must evaluate to 3 and exactly 3).
   B. What is the I/O cost (in pages) of sorting this file? 12000. 2N*3 = 2*2000*3 = 12000
   C. Suppose we want to decrease the I/O cost of sorting this file, but we want to add the minimum number of buffer pages possible. Among the numbers on the answer sheet (1, 5, 10, 50) circle the *smallest* number of additional buffers to add that decreases the I/O cost. We accepted either 5 or 10. A lower IO cost would have a maximum of 2 passes, so B(B-1) = 2000, where B = 40+x. x=5 additional buffers is an approximate answer, whereas 10 additional buffers would be a fully correct answer.

3. [4 points] Given the resources at the top of Question III, answer the following two questions. Do not bother to simplify arithmetic expressions over constants, like 247*(36^3+log(4)).
   A. What is the largest file size (in pages) that we can sort in 3 passes? 40(39)(39). Recall that we can sort B in one pass, B(B-1) in two passes, and B(B-1)^(k-1) in k passes.
   B. What is the smallest file size (in pages) that will require 3 passes to sort? 40(39) + 1. This is the largest file sortable in 2 passes, plus 1.

4. [3 points] Suppose I want to eliminate duplicates from our (unsorted) table of Products, using external hashing. Write down the letters of true statements. (If none are true, write ∅.)
   A. Deduplicating the file using hashing can have a higher IO cost than sorting the file (without deduplicating). True. Consider a file with very, very skewed data, and/or a set of very, very poor hash functions, such that many repartitioning passes are needed.
   B. Deduplicating the file using hashing can have a lower IO cost than sorting the file (without deduplicating). True. Consider an extreme case, where the entire file consists of duplicates - there will only be one tuple to output (and only one pass over the input).
   C. If external hashing recursively partitions on one partition, it will do so on all partitions. False. You only need to recursively partition oversized partitions.

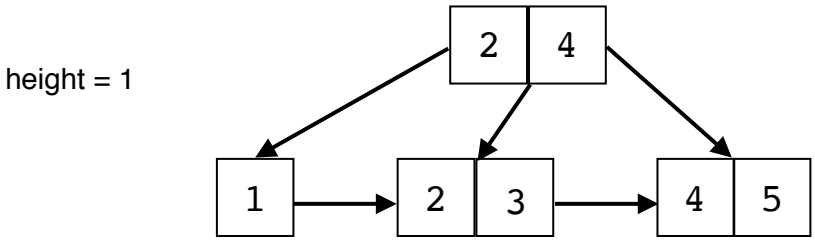
This space left intentionally blank for scratch work.

IV. Indexes and B+ Trees [12 points]

Note: for B+ tree page splits with an odd number of items, assume that the majority of the items is placed on the right-hand page after the split.

1. [4 points] Alphabetically, write down the letters of statements that apply (or write ∅.)
   A. All internal keys in a B+ tree also appear in its leaf nodes. False, a leaf node which has been copied up can be deleted.
   B. The height of a B+ tree increases whenever any node splits. False, height increases when root node splits.
   C. An ISAM index is similar to a B+ tree, but does not allow for insertion of new values. False.
   D. The column(s) we select for our index key must have a unique value for every row in the table. False.
   E. An Alternative 1 index may be either clustered or unclustered. False, it's clustered by design.

2. [5 points] The following B+ Tree has order 1 (max fanout of 3) and each leaf node can hold up to 2 entries. Answer each of the follow questions **independently of each other**.



   A. What value(s) would be in the root node if we were to insert 0? 2, 4
   B. What value(s) would be in the root node if we were to insert 6? 4
   C. Starting with the height 1 tree in the picture above, suppose we start inserting keys 6, 7, 8, … and so on. After inserting what key will the height of the tree become 3? 10

3. [3 points] Assume we are trying to construct a B+ Tree of order 2 (max fanout of 5). Each leaf node can hold up to 4 entries. We insert a total of 16 unique keys via bulk loading, with a fill factor of ¾.
   A. How many leaf nodes will there be? 6
   B. How many internal (non-leaf) nodes will there be? 3
   C. How many internal (non-leaf) nodes do we traverse to do an equality search? 2

V. SQL [17 points]

Consider the following schema:

```
CREATE TABLE Location (
lname TEXT,
areacode INTEGER PRIMARY KEY)
```

```
CREATE TABLE People(
id INTEGER PRIMARY KEY,
pname TEXT,
areacode INTEGER REFERENCES Location)
```

```
CREATE TABLE Calls(
cid INTEGER PRIMARY KEY,
date DATETIME,
from INTEGER REFERENCES People,
to INTEGER REFERENCES People,
duration INTEGER)
```

You should assume that referential integrity is being enforced, and no NULL values appear.

For parts 1 and 2, fill in the blanks in the SQL queries.

1. [7 points] Names of pairs of people who called each other on 2014-12-25

```
SELECT p1.name, p2.name
   FROM People p1, People p2, Calls C
   WHERE C.from = p1.pid
      AND C.to = p2.pid
      AND C.date = '12-25-2014'
```

2. [7 points] Find the location to which the most phone calls have been made, and return its location name as well as the number of calls made to it.

```
SELECT L.lname as name, COUNT(*) as NumCalls
   FROM Location L, People P, Calls C
   WHERE C.areacode = L.areacode
      AND P.id = C.to
GROUP BY  L.areacode
   ORDER BY NumCalls DESC
   LIMIT 1;
```

3. [3 points] On the answer sheet, alphabetically write down letters of the statement(s) that find the name of the location of the person who made the longest call. (If none match, write ∅.)

A.
```
WITH (SELECT P.areacode AS areacode, C.duration AS duration
      FROM calls C, People P
      WHERE C.from =  P.id) AS DC,
SELECT L.lname
FROM DC, Location L
WHERE L.areacode = DC.areacode
ORDER BY DC.duration DESC
LIMIT 1;
```

B.
```
SELECT L.lname
FROM Location L, (SELECT P.areacode AS areacode,
                         C.duration AS duration
                  FROM Calls C, People P
                  WHERE C.from = P.id) AS DC
WHERE L.areacode = DC.areacode
ORDER BY DC.duration ASC;
```

C.
```
WITH (SELECT L.lname AS lname, L.areacode AS LAC, P.id AS pid
      FROM Location L FULL OUTER JOIN People P
           ON L.areacode = P.areacode) AS Names,
SELECT Names.lname
FROM Names, Calls C
WHERE C.cid = Names.pid
ORDER BY C.duration DESC
LIMIT 1;
```

VI. Relational Algebra [8 points]

Consider the schema from the SQL question, but with the relation names shortened:

- Location → L
- People → P
- Calls → C

1. To answer the following two questions, put one relational algebra operator or relation name in each blank:
   a. [3 points] Areacodes to which no call has ever been made.

$\pi_{areacode}\, L \;-\; \pi_{areacode}\,(P \bowtie_{P.id\,=\,C.to} C)$

   b. [3 points] Names of people who live in the location "City of Joe"

$\pi_{name}\,(P \bowtie (\sigma_{pname\,=\,\text{"City of Joe"}}\,(L))$

2. [2 points] The equivalence relation $\equiv$ means that two expressions produce the same results on all possible databases. Write down the letters of the equivalences below that are true. (If none are true, write $\varnothing$.)

A. $\rho(Temp1(1\rightarrow id),\ (\pi_{id}(P) \cup \pi_{from}(C))$
$\rho(Temp2(1\rightarrow id),\ ((\sigma_{id=5}(P) \bowtie_{id=from} C))$

$\sigma_{id=5}(Temp1) \equiv \pi_{id}(Temp2)$

B. $\pi_{areacode}(P) - \pi_{areacode}(\sigma_{areacode=415}(P \bowtie L))$
$\equiv \pi_{areacode}(P) - \pi_{areacode}(\sigma_{areacode=415}(\pi_{areacode}(P) \cap \pi_{areacode}(L)))$

This space left intentionally blank.