CS143 Spring 2016
Midterm Correction

The relation: **warehouse(PartNo, SupplierNo, Price)**
describes the suppliers for each part, along with the price they charge. To cut inventory, the manager of our warehouse wants to eliminate non-competitive suppliers. Competitive suppliers are those that supply at least two parts at a minimum cost (however they might share this minimum with other suppliers). All the others are non-competitive suppliers.

A1 *Write an SQL query to find all non-competitive suppliers.*

```
%competive suppliers
select SupplierNo
       from warehouse as w1
      where w1.Price = (select min(w2.Price)
                        from warehouse as w2
                        where w2.PartNo=w1.PartNO)
group by SupplierNo having count(w1.PartNo) >=2

%for non-competive change >= to <2.
```

1.B Change the definition: competitive suppliers are those who charge a price strictly lower than those of all other suppliers on two or more items. Find the non-competitive suppliers according to the new definition.

```
%competive suppliers
select SupplierNo
         from warehouse as w1
        where w1.Price < ALL (select w2.Price
                                          from warehouse as w2
                                           where w2.PartNo=w1.PartNO and
                                                    w2.SupplierNo <> w1.SupplierNO)
group by SupplierNo having count(w1.PartNo) >=2

%for non-competive change >= to <2.
```

Problem 2: 30 points

Suppose that blocks can hold 100 search keys and 101 pointers. We built a *dense* index organized as a B+ tree on a file of 2 millions records, where the records are placed in blocks that hold 10 records each. Say that the search key for the B+ tree is the candidate key of the relation. Answer the following questions assuming the *worst case* scenario:

A. Compute the blocks used at each level of the B+ tree.

Candidate key: as many pointers are there are records. 2 millions pointers.

Bottom level: 2 millions divided by $50 = 40$ thousands

Next level: 40,000 divided by 51. Take the floor. 784 pointers

Next level: 784 divided by 51. 15 Blocks.

The root.

Total number of blocks $40000 + 784 + 15 + 1$

If this is a sparse index we only need 200,000 pointers. At bottom level 4,000, next level 78. Finally, the root.

B. How many index blocks and file blocks will we have to access to find a record with a given key value if this is a dense index? (Assume that no block is initially in memory, and make the same assumption for the questions that follow.)

Answer: 4+1 blocks

C. Same question as in B, but now assume that our B+ tree is a sparse primary index.

Answer: 3+1 blocks

D. Here too assume that our our B+ tree is a sparse primary index. We now have a range query that is satisfied by 1000 records: how many file blocks will we have to access to retrieve those 1000 record?

Answer: 101 blocks are required to hold 1000 records. So three nodes at the bottom level and two more at the next level. Say around 106 blocks

E. Assume that K was defined as the primary candidate Key in the SQL declarations of R. Can we use a sparse secondary index on the stored R table to speed-up the enforcement of the uniqueness constraint for K?

Answer: Sparse indexes only work as primary indexes. Logical keys..

Problem 3: 20 points Multiple choice questions. Mark the box of every answer that is true.

A. Relations R and S have attributes a and b. Then, which of the properties below is true for the following RA queries:

Q1: $\pi_a(R) \cap \pi_a(S)$ Q2: $\pi_a(R \cap S)$

□ (a) Q1 and Q2 always produce the same answer.

□ (b) The answer to Q1 is always contained ($\subseteq$) in the answer to Q2.

□ (c) The answer to Q2 is always contained ($\subseteq$) in the answer to Q1.

□ (d) None of the above is true.

Answer: the intersection contains the tuples that identical in the first component in the first case, and those are identical in both, in the second case. Thus the second case produces a subset of the first case. The answer is (c)

Problem 3: 20 points Multiple choice questions. Mark the box of every answer that is true.

A. Relations R and S have attributes a and b. Then, which of the properties below is true for the following RA queries:

Q1: $\pi_a(R) \cap \pi_a(S)$ Q2: $\pi_a(R \cap S)$

☐ (a) Q1 and Q2 produce the same answer.

☐ (b) The answer to Q1 is always contained in the answer to Q2.

☐ (c) The answer to Q2 is always contained in the answer to Q1.

☐ (d) Q1 and Q2 produce different answers.

Answer: the intersection contains the tuples that identical in the first component in the first case, and those are identical in both, in the second case. Thus the second case produces a subset of the first case. The answer is (c)

B. When null values are allowed in column R.a or column R.b, the value of the following logic expression in SQL

$$R.a > R.b \text{ AND } R.a < 0 \text{ AND } R.b > 0$$

can be, depending on the tuple considered:

☐(a) only TRUE or FALSE

☐(b) only FALSE or UNKNOWN

☐(c) only TRUE or UNKNOWN

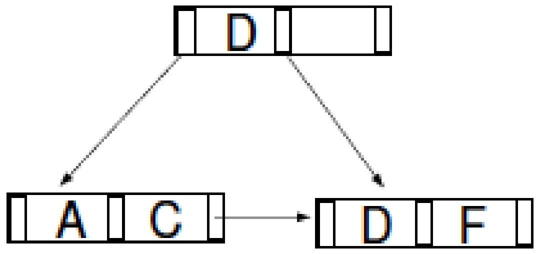
Answer: For tuples without null values this conjunction evaluates to FALSE. But if either the a column or the b column, or both, are null then the comparisons might produce null, and the tree-valued logic AND of nulls is still null. So, the correct answer is (b).

Problem 3: 20 points Assume that we use B+ trees of order n = 3 (n is the maximum number of pointers in a node)

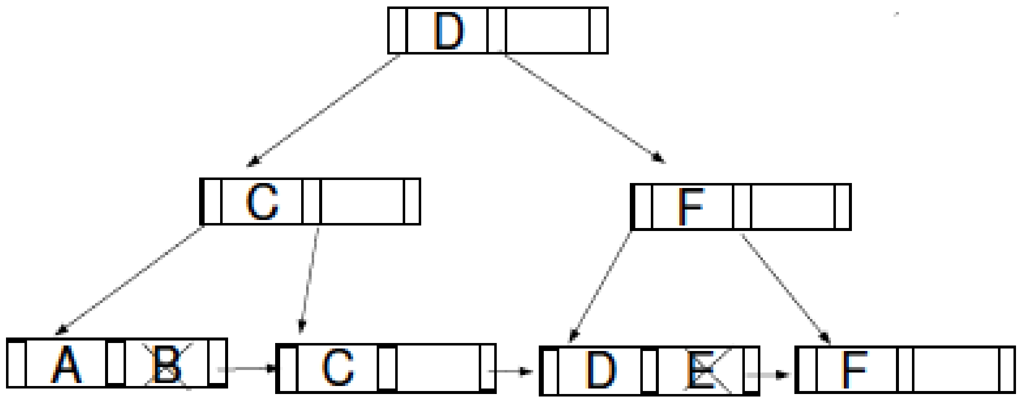
(i) Draw a tree of height 2 with the following keys: A, C, D, F (lexicographically ordered).

(ii) Show how the insertion of two new keys, followed by their deletions, could change this 2-level tree into a 3-level tree with exactly the same keys (i.e., A, C, D, F). Give an example of such keys, and draw the 3-level tree so generated.

Two level Tree:



Insert B and E and then delete them



There are other correct answers.

Extra Credit: 5 points

Overflow buckets are used for both (i) Static Hashing, and (ii) Extensible Hashing, but not in the same way. Explain the kinds of events that cause the creation of new overflow buckets for (i) and (ii): clarify their respective differences and the different actions taken by the hashing module in those cases.

In static hashing: when we try to add a new record to a full block we always create an overflow bucket.

In extensible hashing: when we try to add a new to a full block, if the records in the block have the same key as the one we are adding then we create an overflow block. Otherwise, we extend the hashing by splitting the full page and adjusting the directory accordingly.