CSCI4380 – Database Systems
Spring 2004, Examination I
(100 points + 5 points bonus)

1. (6 points) Consider a relation R(A,B,C,D,E,F). The only thing you know about R is that ABCD and EF are keys. What is the maximum number of keys that R can possibly have (including ABCD and EF)? Explain your calculation.

   Since ABCD and EF are keys, no subset or superset can be a key (recall that a key is minimal). The largest number of keys can be found by add an E or an F to selected subsets of ABCD. The largest number of subsets of ABCD are of length 2, i.e., $\binom{4}{2} = 6$ (namely, AB, AC, AD, BC, BD, CD). To these we can add either an E or an F to obtain a new key. So the total number of keys that are not sub/supersets of ABCD or EF are 12 (namely, ABE, ABF, ACE, ACF, ADE, ADF, BCE, BCF, BDE, BDF, CDE, CDF). Adding the two original keys, we have a maximum of 14 possible keys for the relation.

2. (18 points) Consider the following relational schema (keys are underlined)

   Product(<u>pid</u>, name, price, <u>mfgr</u>)
   Buys(<u>cid, pid</u>)
   Customer(<u>cid</u>, cname, age)

   (a) Write the following query in relational algebra without using the division operator: "Find the names of all customers who have purchased all products that are not manufactured by Sears."

   First let's write the expression for the products not manufactured by Sears.

   $$B : \pi_{pid}(\sigma_{mfgr \neq Sears}(Product))$$

   Now lets find out the names and pid of all customers who buy some product.

   $$A : \pi_{cname,pid}(Buys \bowtie Customer)$$

   The division operator A/B is equivalent to

   $$\pi_{cname}(A) - (\pi_{cname}(\pi_{cname}(A) \times B) - A)$$

   (b) Write the following query in SQL: "Find the names and cids of all customers who have purchased the second most expensive product." You can assume that no two products have the same price.

```
SELECT cname, cid
FROM Customers C, Buys B, Product P
WHERE C.cid = B.cid AND B.pid = P.Pid
      AND P.Price = (SELECT MAX(P2.Price)
                     FROM Product P2
                     WHERE P2.Price < (SELECT MAX(P3.Price)
                                       FROM Product P3)
                    )
```

   (c) Consider the following SQL query:

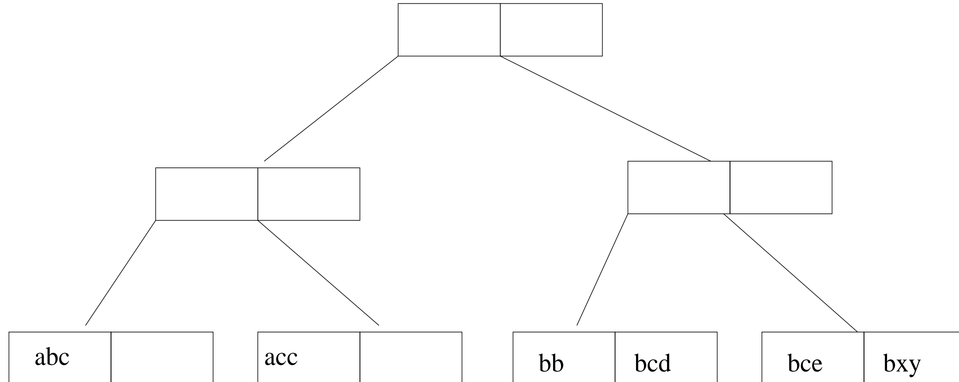
```
SELECT   C.cid
FROM     Customer C, Buys B, Product P
WHERE    C.cid = B.cid and B.pid = P.pid
GROUP BY C.cid
HAVING   count(*) > 100
```

Rewrite the above SQL query without using the "Group By" and "Having" clauses so that the resulting query still produces the same query result. (5 points)

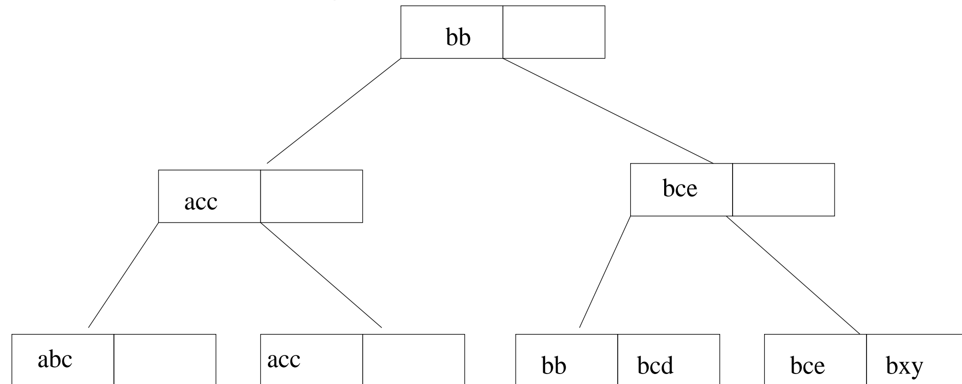
This query asks to find all customers who have bought more than 100 products. Here is how to express it in another way.

```
SELECT C.cid
FROM Customers C
WHERE 100 < (SELECT COUNT(*)
             FROM Customer C2, Buys B, Product P
             WHERE C.cid = C2.cid AND B.cid = C2.cid AND B.pid = P.pid)
```

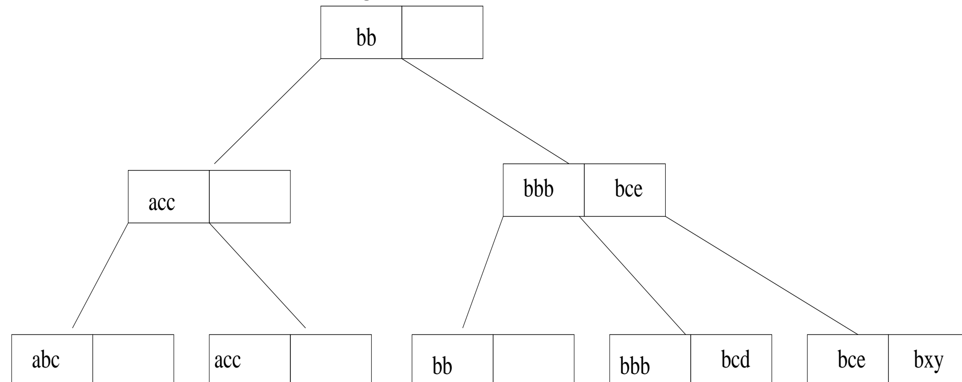
3. (16 points) Consider the B-tree shown:



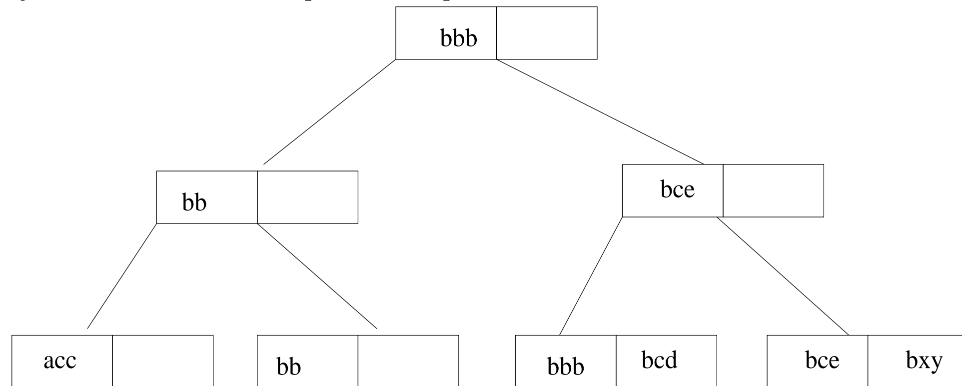
(a) Fill in the internal nodes without adding new keys.



(b) Add the key bbb. Show how the tree changes.



(c) Delete the key abc from the result of previous step. Show the new tree.



4. (12 points) Consider the linear hashing index shown; assume a bucket splits whenever an overflow page is created. Each page can hold at most 4 entries (a '-' denotes empty slot).

```
level = 0, N=4
h1   h0
000  00  [32,20,-,-]  <--Next=0
001  01  [9,25,5,-]
010  10  [10,-,-,-]
011  11  [31,15,7,3]
```

(a) What is the *maximum* number of entries that can be inserted before a bucket split occurs. Explain with an example.

Since a split occurs whenever there is an overflow page created, we can add 6 entries which would fill up the slots, and when we add the seventh entry, it would cause a bucket split.

(b) Show the file after inserting a single record that causes a split.

A split can happen if we add another element to the last bucket, say for example if we add 11 (binary "1011"), shown below.

```
level = 0, N=4
h1   h0
000  00  [32,-,-,-]
001  01  [9,25,5,-]   <--Next=1
010  10  [10,-,-,-]
011  11  [31,15,7,3]-->[11,-,-,-]
100  00  [20,-,-,-]
```

5. (26 points) The "left anti join" operator is a variant of the relational join operator, and is the logical inverse of the relational division operator. The basic idea is to output only those tuples in the left relation that do not join with any tuple in the right relation. For example, consider two relations R(A, B), and S(B, C). Assume that R has the following tuples: (1, 10), (1, 20), (2, 30), (2, 40) and S has the following tuples: (10, 75), (10, 85), (30, 95). R LeftAntiJoin S will produce the following tuples: (1, 20) and (2, 40).

Given the above description of left anti joins, answer the following questions.

(a) How can you adapt the block nested-loops join algorithm to process left anti joins? Describe essential modifications to the data structures and algorithm.

Anti-joins can be easily processed using block nested-loops join. The following pseudo-code explains how.

```
foreach block of B-2 pages of R do
     foreach page of S do
        for all matching tuples r in R-block and s in S-page
            mark r
     foreach tuple r in R-block, output r if unmarked
```

(b) How can you adapt the sort-merge join algorithm to process left anti joins? Describe essential modifications to the data structures and algorithm.

Sort-merge join can similarly be adapted to do anti-joins. The basic idea is that once the relations are sorted, partitions are created, such that values are the same within each partition. Thus we need only to check for a given partition of R, whether there exists a partition of S with the same value. If yes, we can skip the current partition of R, since all tuples in that partition have the same value. If not, then we can output all the tuples in the current partition.
Following the principle of advancing at least once partition of R or S, as in merging, this would cost $|R| + |S|$ page I/Os, assuming partitions fit entirely in buffer.

6. (22 points) Consider the following schema (keys underlined):

Employee(ssn, name, dept)
Project(ssn, pid, name, budget)

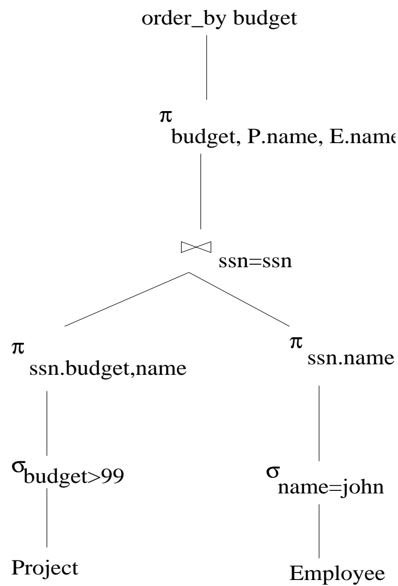
The ssn in project is the id of the employee working on the project. Consider the query:

```
SELECT P.budget, P.name, E.name
FROM Employee E, Project P
WHERE E.ssn = P.ssn AND
    P.budget > 99 AND
    E.name = 'John'
ORDER BY P.budget
```

Also assume the following statistical information:

- 10,000 tuples in Employee
- 20,000 tuples in Project
- 40 tuples/page in each relation
- 10-page buffer
- 1000 different values for E.name
- Domain of budget is integers in the range 1 to 100
- Indexes for Employee:
  - On name: unclustered hash index
  - On ssn: clustered, 3-level B^+ tree
- Indexes for Project:
  - On ssn: unclustered, hash index
  - On budget: clustered, 2-level B^+ tree
- You may assume a 1.2 I/O cost for hash index

(a) Draw the fully pushed query tree



(b) Find the best execution plan. What is the cost?

Select employee on name (equality search): There are on about 10,000/1000 = 10 tuples in the answer. Using the unclustered hash index, we need 1.2 page I/Os to locate the index page, and then need a maximum of 10 page I/Os to retrieve all the 10 tuples (each one can be on a separate page), giving a total of 11.2 pages. From these 10 pages we can extract 10 tuples. which all will fit on 1 page.
Select project on budget (range search): Sine the range of values of budget is 1 to 100, we expect only $100 - 99/100 = 1/100 = 0.01$ fraction of the tuples to be in the answer. Since project has 20,000 tuples, this means we have only 200 tuples in the answer, which would occupy 5 pages. Searching the B-tree index takes another 2 page I/Os. This gives a total of 7 pages.
Since there are only 1 and 5 pages of Employee and Project, respectively, and we have 10 buffer pages, we can do the join completely in memory, using either block nested loops join or sort-merge join. This incurs no additional page I/O. Finally we can project and order on the fly. This gives a total cost of 11.2+7=18.2 pages (not counting output cost).

7. (BONUS: 5 points) Prove mathematically that for block nested loops join it is always better to choose the smaller relation as the outer relation.

Let R and S be the two relations, and let $|R| < |S|$, where $|R|$ denotes the number of pages of R. Let B be the buffer size. The cost of block nested loops join, assuming R to be the outer relation, is given as:

$$|R| + (|S| \times |R|/(B-2)) \tag{1}$$

If S is the outer relation, we have the cost as:

$$|S| + (|R| \times |S|/(B-2)) \tag{2}$$

Since the term $|R||S|/(B-2)$ is the same for both, and since $|R| < |S|$, it is clear that R should be the outer relation.