

CMPSCI 645 — Midterm

April 3, 2006

100 Points

90 Minutes

Problem 1 (7 parts, 30 points total) Queries, query languages, and analysis.

Consider the following relational schema. An employee can work in more than one department.

- Emp(eid, ename, age, salary)
- Works(eid, did)
- Dept(did, dname, budget, managerid)

(a) (6 points) Write an SQL to print the names and ages of each employee who works in both the Hardware department and the Software department.

```
SELECT DISTINCT E.ename, E.age
FROM Emp E, Works W1, Works W2, Dept D1, Dept D2
WHERE E.eid = W1.eid AND W1.did = D1.did AND D1.dname = 'Hardware' AND
      E.eid = W2.eid AND W2.did = D2.did AND D2.dname = 'Software'
```

(b) (1 point) Is the query above expressible as a Conjunctive Query? (Yes or No)

Yes.

- (c) (6 points) Write an SQL query to find the managerids of managers who manage **only** departments with budgets greater than 1 million dollars.

Answer:

```
SELECT DISTINCT D.managerid
FROM Dept D
WHERE 1000000 < ALL (SELECT D2.budget
                    FROM Dept D2
                    WHERE D2.managerid = D.managerid )
```

- (d) (1 point) Is the query above expressible as a Conjunctive Query? (Yes or No)

Answer: No. Not monotonic.

(e) (3 points) Consider the following two SQL queries:

```
Q1: SELECT DISTINCT E.eid
     FROM Emp E, Works W, Dept D
     WHERE E.eid = W.eid AND W.did = D.did AND D.budget = 100000
```

```
Q2: SELECT DISTINCT E.eid
     FROM Emp E, Works W, Dept D1, Dept D2
     WHERE E.eid = W.eid AND W.did = D1.did AND W.did = D2.did
           D1.budget > 50000 AND D2.budget < 200000
```

State which of the following holds, and **briefly explain** your answer:

- i. $Q_1 \subset Q_2$
- ii. $Q_2 \subset Q_1$
- iii. Q_1, Q_2 are equivalent
- iv. None of the above.

Answer: (i) holds.

(f) (6 points) Associate each query language class listed below with an equivalent class from: the conjunctive queries (**CQ**), the first order queries (**FO**), or the recursive queries (**Recursive**).

- i. Unions of conjunctive queries with negation **FO**
- ii. The relational algebra restricted to the operators \times, σ, Π **CQ**
- iii. Datalog queries (no negation) **Recursive**
- iv. Queries expressible as single SELECT DISTINCT - FROM - WHERE blocks. **CQ**
- v. Non-recursive datalog queries (no negation) **Union of CQs (CQ accepted as right answer)**
- vi. The relational algebra **FO**

(g) (7 points) A Boolean tree has leaves labeled with 0 or 1, and internal nodes which are operators AND, OR, **NOT**. The root of a boolean tree returns 0 or 1. We can represent a Boolean tree using the following stored (EDB) predicates:

- Leaf0(x): a unary table containing the identifiers of each leaf labeled 0. (Alternatively, you can think of Leaf0(x) as a logical predicate that is true of node x if x is a leaf labeled 0.)
- Leaf1(x): a unary table containing the identifiers of each leaf labeled 1.
- And(x, y_1, y_2): a ternary table containing tuples (x, y_1, y_2) if x is an AND node whose children are nodes y_1 and y_2 .
- Or(x, y_1, y_2): a ternary table containing tuples (x, y_1, y_2) if x is an OR node whose children are nodes y_1 and y_2 .
- Not(x, y): a binary table containing tuples (x, y) if x is a NOT node whose child is node y .
- Root(x): a unary table containing the root node.

Write a datalog query that computes the boolean relation Answer() which is true if and only if the root node returns 1. Hint: your program should compute IDB relations One(x) and Zero(x), in addition to Answer().

```

Zero(x)    :- Leaf0(x)
Zero(x)    :- AND(x, y1, y2), Zero(y1)
Zero(x)    :- AND(x, y1, y2), Zero(y2)
Zero(x)    :- OR(x, y1, y2), Zero(y1), Zero(y2)
Zero(x)    :- Not(x,y), One(y)
One(x)     :- Leaf1(x)
One(x)     :- AND(x, y1, y2), One(y1), One(y2)
One(x)     :- OR(x, y1, y2), One(y1)
One(x)     :- OR(x, y1, y2), One(y2)
One(x)     :- Not(x,y), Zero(y)
Answer()   :- Root(x), One(x)

```

Problem 2 (5 parts, 20 points total) Normalization and redundancy

(a) (4 points) Consider the following table:

A	B	C
a11	b22	c55
a11	b22	c66
a11	b33	c77
a22	b44	c66

Indicate which of the functional dependencies hold for this instance. Answer Yes or No for each part:

- $B \rightarrow A$ Yes.
- $B \rightarrow C$ No.
- $A, B \rightarrow C$ No.
- $B, C \rightarrow A$ Yes.

- (b) (6 points) For $R(A,B,C,D)$ and functional dependencies $AB \rightarrow C$, $AB \rightarrow D$, $C \rightarrow A$, $D \rightarrow B$ state the strongest normal form the relation is in. Briefly explain your answer.

Solution: 3NF

- (c) (2 points) Is redundancy (with respect to functional dependencies) possible for this relation? Answer Yes or No.

Solution: Yes, redundancy is possible for relations in 3NF.

- (d) (6 points) For $R(A,B,C,D,E)$ and functional dependencies $A \rightarrow BC$, $BC \rightarrow E$, and $E \rightarrow DA$, state the strongest normal form the relation is in. Briefly explain your answer.

Solution: BCNF

- (e) (2 points) Is redundancy (with respect to functional dependencies) possible for this relation? Answer Yes or No.

Solution: No, redundancy is not possible for relations in BCNF.

Problem 3 (10 points total) State if the following statements are TRUE or FALSE.

- (a) Suppose that a randomly chosen block is being fetched from disk. The overall access delay per byte decreases as the block size increases.
- (b) The LRU (Least Recently Used) buffer replacement strategy is a good strategy for repeated sequential scans of the same file.
- (c) Insertion order into a B+ Tree will effect the tree's end structure.
- (d) One way to handle duplicate keys in an unclustered index is to maintain a pointer to only the first occurrence of the key in the file.
- (e) B+ trees are typically more efficient than linear or extensible hashing for all types of queries.
- (f) A B+Tree on $\langle \text{age, salary, department} \rangle$ can be used to answer a selection condition "age=20 AND department = 'CS' ".
- (g) A hash index on $\langle \text{age, salary} \rangle$ can be used to answer a selection condition "age=20 AND salary > 5000".
- (h) Since there is no downside to having indexes, we may as well build an index on every column of every relation to speed up as many queries as possible.
- (i) $\sigma_p(E_1 - E_2) = \sigma_p(E_1) - E_2$
- (j) $\pi_s(E_1 - E_2) = \pi_s(E_1) - \pi_s(E_2)$

Solution: a) T, b) F, c) T, d) F, e) F, f) T/F, g) F, h) F, i) T, j) F.

Problem 4 (4 parts, 25 points total): Indexes and File Organization

(a) (6 points) **Circle** the basic file organization (heap, sorted, or hash) that is best for a large file where the most frequent operations are as follows (answer each separately – no explanation needed):

i. Search for records based on a range of field values.

HEAP SORTED HASH

ii. Perform inserts and scans where the order of records does not matter.

HEAP SORTED HASH

iii. Search for a record based on a particular field value.

HEAP SORTED HASH

1) SORTED 2) HEAP 3) HASH

(b) (8 points) Create a B+tree where each node can hold at most 4 pointers and 3 keys when the following keys are inserted in the following order:

1, 10, 2, 11, 3, 4, 8, 5, 7, 6

Solution:

Root: 7

Level 1: (3, 5) (10)

Level 2: (1, 2) (3,4) (5,6) (7,8) (10,11)

c) (6 points) Assume that a B+tree can hold at most n keys in each node and the height of the tree is h . Also assume that there is a 50% minimum occupancy requirement for each node, except the root. What are the **minimum** and **maximum** numbers of keys can this B+tree hold?

Assume that the root holds r keys and each other node holds m keys.

Level 0: r
 Level 1: $(r+1)m$
 Level 2: $(r+1)(m+1)m$
 ...
Level h : $(r+1)(m+1)^{h-1}m$

Total $= r + (r+1)(m+1)^0m + (r+1)(m+1)^1m + \dots + (r+1)(m+1)^{h-1}m$
 $= r + (r+1)m \left[\frac{(m+1)^h - 1}{m} \right]$
 $= r + (r+1) [(m+1)^h - 1]$

Maximum: $r = n, m = n$, so

Level h : $(n+1)^h n$
 Total $= n + (n+1) [(n+1)^h - 1]$
 $= (n+1)^{h+1} - 1$
 $= O((n+1)^{h+1})$

Minimum: $r = 1, m = \lceil n/2 \rceil$, so

Level h : $2(\lceil n/2 \rceil + 1)^{h-1} \lceil n/2 \rceil$
 Total $= 1 + 2[(\lceil n/2 \rceil + 1)^h - 1]$
 $= O((\lceil n/2 \rceil + 1)^h)$

Note: in this question, answers for either level h or the total can be accepted.

(c)

- (d) (5 points) Suppose that you have a file that is already sorted in key order and you want to construct a clustered B+ tree index on this file using (key, RID) pairs for data entries. A simple way to accomplish this is to create an empty B+tree, and then sequentially scan the file, inserting an index entry for each record using the normal B+tree insertion routine. What **storage utilization** problem does this approach have? **Briefly** describe a solution that would solve the problem.

Solution: Storage utilization problem: Most leaf nodes are half full as a result of inserting sorted key values and splitting nodes at the leaf level in a way that each of the two nodes is half full. So, utilization is roughly 50

[Associated performance problems: Because the tree is large, the performance is not good. Another performance problem is that to insert each key, the B+ tree is traversed from root to the leaf.]

One of the two solutions would work:

- One is to change the splitting method. When a leaf node is split, do not move half key values to the newly created node. Just move the last key to the new node so that the left node is still full.
- Bulk loading. That is to build B+ tree bottom up from sorted key values.

Problem 5 (3 parts, 15 points total) Query Optimization

Consider the following schema:

- Sailors(sid, sname, rating, age)
- Boats(bid, bname, size)
- Reserves(sid, bid, day)

Reserves.sid is a foreign key to Sailors and Reserves.bid is a foreign key to Boats.bid.

We are given the following information about the database:

- Sailors contains 1000 records with 20 records per page.
- Boats contains 100 records with 10 records per page.
- Reserves contains 10,000 records with 40 records per page.
- There are 100 values for Reserves.sid.
- There are 50 values for Reserves.bid.
- There are 1000 values for Reserves.day

In the following queries, assume that a System R style optimizer is used.

Consider Query 1:

```
SELECT S.sid, S.sname, B.bname
FROM   Sailors S, Reserves R, Boats B
WHERE  S.sid = R.sid AND R.bid = B.bid AND R.day = 'July 4, 2003';
```

a) (6 points) Assuming uniform distribution of values and column independence, estimate the number of tuples returned by this query.

Solutions: The maximum cardinality this query is $|R| \times |S| \times |B| = 10^9$. (1 pt) Reduction Factors: 1/1000 for $R.day = 'July 4, 2003'$, (1 pt) 1/100 for $R.bid = B.bid$, (1.5 pts) 1/1000 for $S.sid = R.sid$ (1.5 pts) So number of tuples returned is $(1/1000)(1/100)(1/1000)(10^9) = 10$ (1 pt)

Consider Query 2:

```
SELECT S.sid, S.sname, B.bname
FROM Sailors S, Reserves R, Boats B
WHERE S.sid = R.sid AND R.bid = B.bid
```

b) (4 points) Draw all possible left-deep query plans for this query:

$$(R \bowtie S) \bowtie B, (S \bowtie R) \bowtie B, (R \bowtie B) \bowtie S, (B \bowtie R) \bowtie S$$

Note that we cannot join S and B since that would result in a cross product.

c) (5 points) Assume that the first join in a query plan (the one at the bottom of the tree) is $R \bowtie S$. Assume that you have 51 pages of memory. There are no indexes, so indexed nested loops join is not an option. What join algorithm would work best? Briefly explain your answer.

$|R| = 250, |S| = 50, |B| = 10$.

R join S:

- Page-oriented nested loops join: $|R| + |R| \times |S| = 250 + 250 \times 50 = 12570$
- Block nested loops join: $|R| + \lceil |R|/49 \rceil \times |S| = 250 + 6 \times 50 = 550$
- Sort-merge join: note that $250 < (\text{num buffers})^2$ so cost is $3(|R| + |S|) = 3(250 + 50) = 900$
- Hash join: we can use hash join since each of the R partitions from phase one fit into memory. So cost is $3(|R| + |S|) = 3(250 + 50) = 900$.
- **Cheapest: Block nested loops**