# CSCI 6333/6315 Database

**Final Exam, Spring 2020**

Name: \_\_Ulvi Bajarani\_\_\_\_\_\_\_\_\_

ID: \_\_20539914\_\_\_\_\_\_\_\_\_\_\_\_

***Note:*** *Please type your answers. If you need to draw a figure, you may draw on a paper, take a picture, and then cut and paste the figure to your file.*

**Problem 1** [20 points]. Consider the schema and the following set of functional dependencies holds on

1. Compute .
2. Prove that the decomposition and of

is a lossless-join decomposition.

**Answer 1.**

1. A+ = {A, B, C, D} due to A → BCD;

A+ = {A, B, C, D, E} due to BC → DE;

The answer is A+ = {A, B, C, D, E}.

1. the lossless-join decomposition conditions are:

R1 ∪ R2 = R

# R1 ∩ R2 ≠ ∅

R1 ⋈ R2 → R1 or R1 ⋈ R2 → R2.

Since R1 ∪ R2 = R,

R1 ∩ R2 = {B, F} ≠ ∅,

R1 ⋈ R2 → R1 due to (BF)+ = {A, B, C, D, E, F} (even R1 ⋈ R2 → R2), we prove that the decomposition is the lossless-join.

**Problem 2**. [20 points] Let the relations and have the following properties: has 30,000 tuples, has 45,000 tuples, 25 tuples of fit on one block, and 30 tuples of fit on one block. Estimate the number of block accesses required, using each of the following join strategies for to determine the efficient loop structure, that is, which relation shall be used for the outer loop (and the other is for the inner loop).

1. Nested-loop join
2. Block nested-loop join

**Answer 2.**

br1 = 30000/25 = 1200

br2 = 45000/30 = 1500

1. The best case for Nested-loop join requires br1 + br2 + 2 seeks = 1200 + 1500 + 2 seeks = 2700 disk transfers + 2 disk seeks = 2702 block accesses.

The worst case requires

either

br1 + r1 = 1200 + 30000 = 31200 seeks.

r1 \* br2 + br1 = 30000 \* 1500 + 1200 = 45001200 block transfers.

Total disk accesses = 45001200 + 31200 = 45032400

or

br2 + r2 = 1500 + 45000 = 46500 seeks.

r2 \* br1 + br2 = 45000 \* 1200 + 1500 = 54001500 block transfers.

Total disk accesses = 54001500 + 46500 = 54048000

1. The best case for Block nested-loop join requires br1 + br2 + 2 seeks = 1200 + 1500 + 2 seeks = 2700 disk transfers + 2 disk seeks = 2702 block accesses.

The worst case for Nested-loop join requires

either

2 \* br1 = 2 \* 1200 = 2400 seeks.

br1 \* br2 + br1 = 1200 \* 1500 + 1200 = 1801200‬

Total disk accesses = 1801200 + 2400 = 1803600

or

2 \* br2 = 2 \* 1500 = 3000 seeks.

br2 \* br1 + br2 = 1500 \* 1200 + 1500 = 1801500‬

Total disk accesses = 1801500 + 3000 = 1804500

While both loops give the same result in the best case, b) is much efficient in the worst case.

**Problem 3**. [10 points] Give two concrete examples to show that there are schedules that are possible under the two-phase locking protocol, but are not possible under the timestamp protocol, and ***vice versa***.

**Answer 3.**

1. The schedule with a 2PL protocol which is not possible in a timestamp protocol, because of the step 5, where TS(T0) < W-Timestamp(B).

|  |  |
| --- | --- |
| T0 | T1 |
| Lock-S(A); | Lock-X(B); |
| Read(A); | Write(B); |
|  | Unlock(B); |
| Lock-S(B); |  |
| Write(B); |  |
| Unlock(A); |  |
| Unlock(B); |  |

1. The schedule with a timestamp protocol which is not possible in 2PL, because the unlock on A should be occurred between steps 2 and 3, and the unlock on B should be occurred between steps 4 and 5.

|  |  |  |
| --- | --- | --- |
| T0 | T1 | T2 |
| Write(A); |  | Write(A); |
|  | Write(A); |  |
|  |  |  |
| Write(B); |  |  |
|  | Write(B); |  |

**Problem 4**. [10 points] When the system recovers from a crash, it constructs an undo-list and a redo-list. Explain why log records for transactions on the undo list must be processed in reverse order, while those log records for transactions on the redo-list are processed in a forward direction.

**Answer 4.**

If either the log records for transactions on the undo-list is processed in a normal order or the log records for transactions on redo-list is processed in a reverse order, it will give an erroneous values on the data that was updated several times.

**Problem 5**. [10 points] Let be a relation and a set of functional dependencies of . |R|= n and |F= m.

1. What is the time complexity to check whether R is in BCNF?
2. What is the time complexity to check whether R is in 3NF?

**Answer 5.**

1. The problem has O(m2n2) time complexity, because the computing of attribute closure has O(mn2) time complexity.
2. There is no determined time, since the problem is NP-hard, because it requires to check any attribute A from β-α in all functional dependencies α → β , which is also an NP-hard problem.

**Problem 6**. [10 points] A disk block has 1024 bytes, and both a pointer and a search key are of 4 bytes each. Now, consider that we build a -tree index with a disk block to store every tree node. Assume that the index tree has a height of 4. Estimate the number of unique search key values that can be represented by the index tree.

**Answer 6.**

The worst case.

If we assume pointers as n, we should have [n/2]4 internal nodes (because the height of tree is 4), and [n/2]4 \* [((n+1)/2)-1] search key values, due to the fact that each leaf node should have at least (n-1)/2 search key values. Knowing that internal pointers are at least [(n+1)/2]4 internal pointers, and every leaf node has [n/2]4 \* [((n+1)/2)] pointers, we should calculate:

4\*( [n/2]4 \* [((n+1)/2)-1] ) + 4\*( [(n+1)/2]4 + [n/2]4 \* [((n+1)/2)] ) ≤ 1024 bytes

The maximum n is 4. Therefore, we might fit only [4/2]4 \* [((4+1)/2)-1] = 12 key values and [(4+1)/2]4 + [4/2]4 \* [((4+1)/2)] = 39.0625 + 39.0625 \* 2.5 = 136.71875 ≈ 136 pointers.

The best case.

In the best case, when we have [n]4 internal nodes and each leaf node has (n-1) search key values, we have most number of search keys [n]4 \* (n+1), [(n+1)/2]4 internal pointers and [n]4 \* [(n+1)] leaf pointers. Therefore, we should calculate

4\*( n4 \* (n+1) ) + 4\*( (n+1)4 + [n]4 \* [(n+1)] ) ≤ 1024 bytes

The maximum n is 2. Therefore, we might fit only [2]4 \* [(2+1)] = 48 key values and ( (2+1)4 + [2]4 \* [(2+1)] ) = 81 + 16 \* 3 = 129 pointers.

The answer is between 12 search key values and 136 pointers (the worst case) and 48 key values and 129 pointers (the best case).

**Problem 7** [20 points] Show that the wound-wait strategy will prevent

1. Deadlock, and
2. Starvation.

**Answer 7.**

1. To check the deadlock in the wound-wait strategy, we should check if it creates any cycles. Since that T0 , T1, … Tn has the times α0, α1, … αn , and α1, < α2, … < αn , to create the cycle, it should be α0, < α1, … < α1 < α0 , which is the contradiction. So, there couldn’t be any deadlock.
2. Since that in the wound-wait every transaction has own counter and the conflicting step in the younger transaction will be executed, it prevents the starvation.