ICCS240: FINAL EXAM

Kriangsak Thuiprakhon: 6080163
kriangsak.thi@student.mahidol.edu

1 Multiple Choices

1. False
2. False
3. True
4. True
5. False
6. True
7. True
8. False
9. False
10. True
11. False
12. False
13. True
14. False
15. True
16. False
17. True
18. True
19. False
20. True

2 SQL and Relational Algebra

1. Given a database schemata:Product(pid,price),Order(oid,customer,pid,quantity). Write in SQL to find a list all products along with its total ordered quantities in descending order of the amount.

   SQL:

```
SELECT pid, sum(quantity) AS sum  FROM orders
GROUP BY pid  ORDER BY sum DESC;
```

2. Write the following relational algebra expression in SQL, for relation R(a,b,c) and S(a,b,c): $\pi_{a,b}(\delta_{a<0}(R)) - \pi_{a,b}(S)$

   SQL:

```
SELECT a.,b FROM R WHERE R.a <0 AND NOT EXISTS (SELECT a,b FROM S);
```

3. SELECT a, d FROM R, S WHERE R.a > 240 and R.b = S.c. RELATIONAL ALGEBRA:

$$\Pi_{a,d}(\delta_{a>240}(R) \bowtie_{b=c} (S))$$

3 Design Theory

1. Given relation $R(A, B, C, D, E, G)$ and the set of functional dependencies $\mathcal{F} : A \to CD, AB \to DE, AD \to G, BD \to E, D \to EG, E \to G$ Find a minimal basis of F.

   ANSWER :

   1 Rewrite the FD into those with only one attribute on RHS. We obtain:

$$\begin{aligned} A &\to C \\ A &\to D \\ AB &\to D \\ AD &\to G \\ BD &\to E \\ D &\to E \\ D &\to G \\ E &\to G \\ AB &\to B \end{aligned}$$

   2 Remove trivial FDs (those where the RHS is also in the LHS). We obtain: In this casem noting can be removed

   3 Minimize LHS of each FD. We obtain:

$$\begin{aligned} A &\to C \\ A &\to D \\ A &\to D \\ D &\to G \\ D &\to E \\ D &\to E \\ D &\to G \\ E &\to G \\ A &\to B \end{aligned}$$

4 Remove redundant FDs (those that are implied by others). We obtain:

$$A \to C$$
$$D \to E$$
$$E \to G$$
$$A \to A$$

2. Assume a relation R(A,B) has no NULL values. Write a SQL query that can check whether a functional dependency A ! B holds in R. Briefly explain why your query works.

   ANSWER :

   ```
   SELECT * FROM R r1, R r2 WHERE r1.A=r2.A AND r1.B <> r2.B;
   ```

   This query should return an empty table if A —> B holds. Why does this work? This is because: (first, select A iff $r1.A = r2.A$ from the cross product)—AND— (second, B iff $r1.B <> r2.B$) this means that for every A in R, it is mapped strictly one-to-one to B in R. This is why the resulting table is empty if $A \to B$holds.

3. Consider a relational R(A,B,C) having a functional dependency A ! B. If C is a candidate key for R, determine if R could be in BCNF (YES/NO). If YES, state the condition when/where R could be in BCNF. Otherwise, explain why R cannot be in BCNF.

   ANSWER :

   No, because if C were to be a candidate key, FD A—> B has violated the condition to be in BCNF, that is ,every determinant has to be a candidate key. In this case, A is not a candidate key. Hence, R is not in BCNF

4 Storage and Indexing

1. Consider a relationR(A,B) stored as a randomly ordered file. The only index on R is an *unclustered* index on A. Suppose you want to retrieve all records where $A > 0$. Explain whether using the index is always the best alternatives. What about just simply performing file scanning instead of using the index?

   ANSWER :

   Per the nature of query, we can see that it works in the way that it will filter every other elements that are at most zero. unclustered index is used to optimize filtering queries. One application of unclustered index is filtered index. A filtered index is an optimized nonclustered index especially suited to cover queries that select from a well-defined subset of data. It uses a filter predicate to index a portion of rows in the table. A well-designed filtered index can improve query performance as well as reduce index maintenance and storage costs compared with full-table indexes. The good thing about it is that creating a filtered index can reduce disk storage for nonclustered indexes when a full-table index is not necessary. You can replace a full-table nonclustered index with multiple filtered indexes without significantly increasing the storage requirements.

   Performing a scan over A without indexing will result in a higher time compextity.

2. Consider an application that required user to register their names. Assume at this moment the following users have registered their name in the following order (their age is in the parentheses):

   $alice(10), bob(20), carol(19), david(30), eve(15), frank(60), alex(10), chris(31), diana(99)$

It is expected to have many more users in the future. The application tends to retrieve some (sub) lists of users ordered by names, and sometimes ordered by their ages. What index(es) should be built to help facilitate or improve the application query performance? Show how all of your proposed index(es) looks like. You must also specify any hash, comparison function or any related function you need to build your index(es).

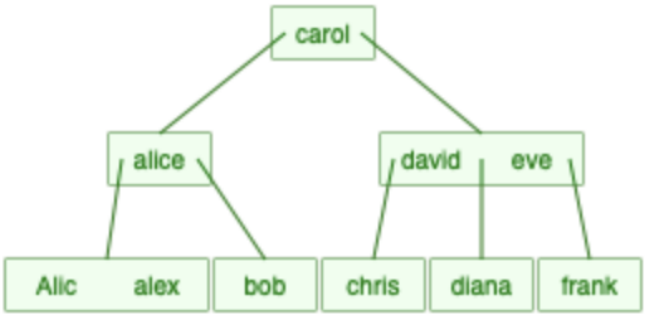
ANSWER

To improve upon performances, I propose clustered index on name attribute and *unclustered* dynamic hashed-based index for the age attribute, specifically, a B+ tree. For the first part, I am using a clustered index so that it makes it easier to perform (sub) lists-of-users-ordered-by-names type of queries. With a B+ tree of degree k, this will be beneficial when querying for lists of users ordered by age.

we can create a clustered index using: Assume that the application has database: $app(namevarchar, ageint)$.

```
CREATE UNIQUE CLUSTERED INDEX nameIndex ON app (name)
```

If we use a B tree the result may look like this:

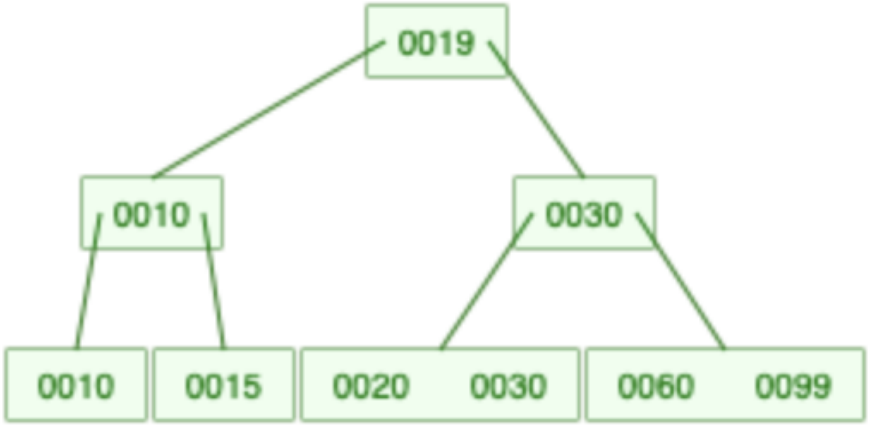


This will index name records in an alphabetical order.

for the age indexing, we can use a similar SQL query to create an index by:

```
CREATE UNIQUE INDEX idxAge ON app ( age ASC);
```

This may result in a tree looking like the following:



5 Transaction and Concurrency Control

1. Consider the actions R(X); W(X) taken by transaction T 1 on database object X. Give an example of another transaction T2 that, if run concurrently to T1 without some form of concurrency control, could interfere with T 1. Briefly explain why.

   ANSWER

   $T_1 : R(X); W(X)$
   $T_2 : W(X); R(X)$
   This causes an interference of in the case that the second transaction performs R(X) first and the first transaction has not yet written , T2 will abort. The value read by T1 would be invalid and the abort would be cascaded to T1 (according to some random website...).

2. Explain what it means for a schedule to avoid-cascading-aborts.

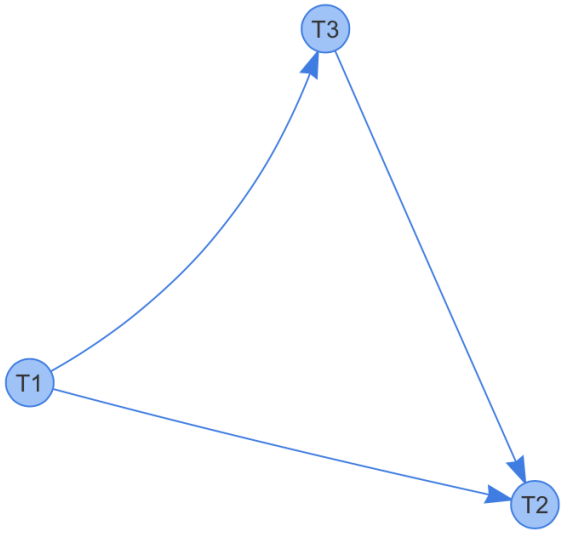
   ANSWER

   A schedule is said to avoid cascading aborts if whenever a transaction reads an element, the transaction that has last written it has already committed. That is, cascadeless Schedule avoids cascading aborts/rollbacks (ACA) Schedules in which transactions read values only after all transactions whose changes they are going to read commit are called cascadeless schedules. Avoids that a single transaction abort leads to a series of transaction rollbacks. A strategy to prevent cascading aborts is to disallow a transaction from reading uncommitted changes from another transaction in the same schedule.
   In other words, if some transaction T_j wants to read value updated or written by some other transaction T_i, then the commit of $T_j$must read it after the commit of T_i.

3. (i) Given a schedule:

   $$R2(C); R1(B); R1(A); W1(B); R3(A); W3(B); R3(C); W2(B)$$

   Please answer the following questions. Draw a precedence graph of the above schedule.
   ANSWER



(ii) Is the schedule conflict serializable? If so, write YES and show the order in which the transaction would have to run in the equivalent serial schedule. Otherwise, write NO. (3pt)

ANSWER

Since the graph is acyclic, the schedule is conflict serializable In Topological Sort, we first select the node with indegree 0, which is T1. This would be followed by T3 and T2.
Hence, the given schedule is conflict serializable since it is conflict equivalent to the serial schedule T1 T3 T2.

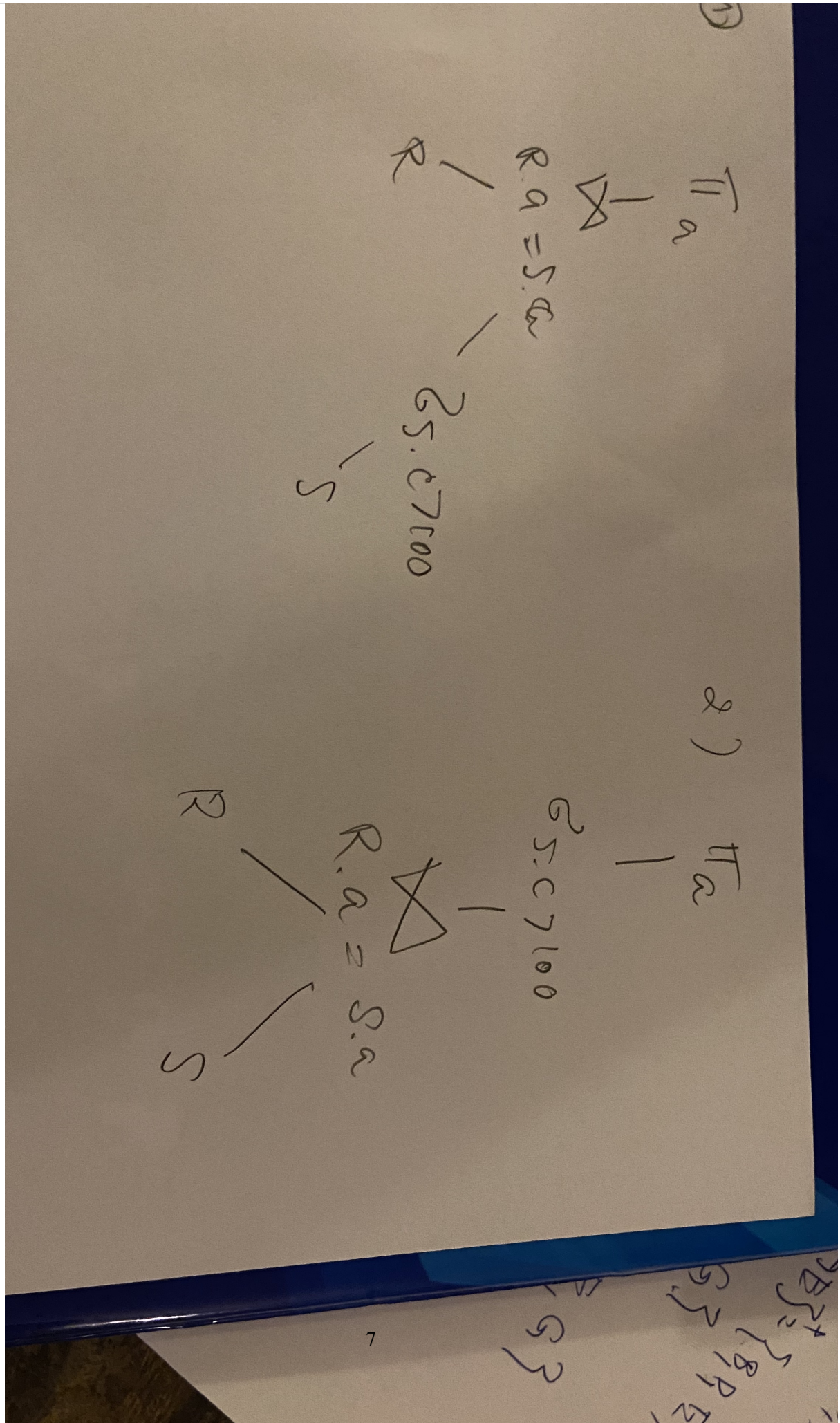
4. No, only one transaction will lock A. Assume without loss of generality, take transaction, let's just take the first transaction for simplicity. From what we learned, a transaction is able to work on something of and only if other transactions stop before being able to lock B. now, the T1 can perform actions on B while locking it, then committing to release A and B. This then causes no deadlocks.

5. i dont know

6 Query Execution and Optimization

1. plans

Kriangsak Thuiprakhon: 6080163
kriangsak.thi@student.mahidol.edu



2. I DON'T KNOW

3. a) B(R) = 10 I/O b) clustered: 10/10 = 1 , unclustered: 2000/10 = 200 c) assume LS, B(R)+B(S)= 10 +80 = 90 d) B(R)= 10 e) 5B(R)+5B(S) = 50 + 400 = 450 f) 10 if no indexing g) $B(R) + 5(B(S) = 410$ h)$B(R) + 5B(S)/90 - B(R) = 12$

7 Extra Credit

7.1 EXAM I

7.2 EXAM II

First find $h(T_1)$ and $h(T_2)$ in $O(h(T_1)+h(T_2))$ time.
(*) Note that for t=2, the minimum keys limitation always holds.

Case a: h(T1)=h(T2):

a. $k \leftarrow \frac{\text{maxKey}(T_1) + \text{minKey}(T_2)}{2}$
b. create a new root with the key k (with null record pointer)
c. T_1 will be the left sub-tree of k and T_2 will be the right sub-tree.
d. Btree-Delete (k)

Case b: $h(T_1) \neq h(T_2)$:
Without loss of generality assume $h(T_1) > h(T_2)$.

1. $k \leftarrow \frac{\text{maxKey}(T_1) + \text{minKey}(T_2)}{2}$
2. Like in Btree-Insert, go down on the rightmost path in T_1 (k is larger than all the keys in T_1), while splitting all full nodes. Stop at the node y, such that $h(y) = h(T_2)+1$. y is not full, i.e., has 2 keys at most.
3. Add k as the largest key in y
4. T_2 will be the right sub-tree of k in the node y.
5. Btree-Delete(k)

Running time: $O(h(T_1)+h(T_2))$

4