CS127 Homework #6

Due: November 27th, 2017 2:59 P.M.

Handing In

Upload your homework to Gradescope.
Please write your Banner ID on your submission. Do not write your name on the submission.

Warmup #1

Assume that the following 2 plans answer the same query:
Plan A:

$$(\sigma_{att=3}A) \bowtie B$$

Plan B:

$$\sigma_{att=3}(A \bowtie B)$$

Answer the following questions:

1. If you had no knowledge of the data which one would you pick? Why?

   Plan A. The Join will be done on smaller tables and join operation are more expensive than selection

2. Is there a situation in which you would prefer the other plan? Why?

   If there are going to be no elements in the join between A and B and all the element in A had $A.att = 3$ than I would prefer plan B. It doesn't make a huge difference (since selection are cheap), but it saves some time.

Warmup #2

Consider the relations $R(A, B), S(B, C)$ and $T(C, D)$, such that their sizes in tuples are respectively $3 * 10^4$, $2 * 10^5$ and 10^4.
Moreover, consider the query:

$$\sigma_{R.A<40}(R \bowtie S \bowtie T)$$

Estimate the size of the size of the resulting relation given the following 3 pieces of information:

1. Selectivity of $R.B = S.B$ is $\frac{1}{3}$
2. Selectivity of $S.C = T.C$ is $\frac{1}{10}$
3. Selectivity of $R.A < 40$ is $\frac{1}{2}$

Size $= 3 * 10^4 * 2 * 10^5 * 10^4 * \frac{1}{3} * \frac{1}{10} * \frac{1}{2} = 10^{12}$
Why? Consider doing the crossproduct between the 3 relations and than using the given selectivities to estimate how many tuples would be selected.

Warmup #3

List the ACID properties. Explain the usefulness of each.

- Atomicity: Either all operations of the transaction are reflected properly in the database, or none are.
- Consistency: Execution of a transaction in isolation (that is, with no other transaction executing concurrently) preserves the consistency of the database.
- Isolation: Even though multiple transactions may execute concurrently, the system guarantees that, for every pair of transactions Ti and Tj , it appears to Ti that either Tj finished execution before Ti started or Tj started execution after Ti finished. Thus, each transaction is unaware of other transactions executing concurrently in the system.
- Durability: After a transaction completes successfully, the changes it has made to the database persist, even if there are system failures.

Warmup #4

Assume there is a database with 2 objects A and B and consider two transactions $T1$ and $T2$.
Give an example an example of schedule composed by reads and writes of $T1$ and $T2$ that result in a read-write conflict.
T2:R(X), T2:R(Y), T1:R(X), T1:R(Y), T1:W(X), T2:R(X),...
T2 will get an unrepeatable read on X.

Problem 5 (To Be Graded)

Consider the following (incomplete) schedule S:

T1	T2	T3
R(X)		
R(Y)		
W(X)		
	R(Y)	
		W(Y)
W(X)		
	R(Y)	

1. Can you determine the serializability graph for this schedule? Assuming that all three transaction eventually commit show the serializability graph.
2. For each of the following, modify S to create a complete schedule that satisfies the stated condition. If a modification is not possible, explain briefly. If it is possible, use the smallest possible number of actions (read, write commit, or abort). Feel free to add commits or aborts to make the schedule easier to read.
   (a) Resulting schedule avoids cascading aborts but is not recoverable
   (b) Resulting schedule is recoverable
   (c) Resulting schedule is conflict-serializable

   A) Recoverable Schedule For each pair of transaction Ti and Tj, if Tj read an object previously written by Ti, Tj commits after Ti Commits.

   Avoids-cascading-abort schedule: For each pair of transaction Ti and Tj, if Tj reads an object previously written by Ti, Ti commits before the read operation of Tj. Thus not possible!

   B) Consider the following (incomplete) schedule S:

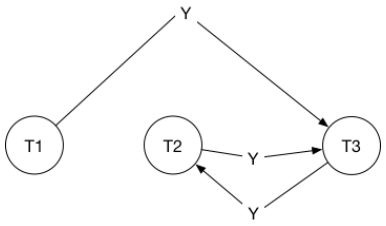

Figure 1: a

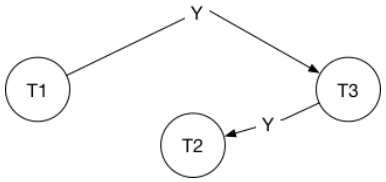

Figure 2: b

T1	T2	T3
R(X)		
R(Y)		
W(X)		
W(X)		
Commit		
		W(Y)
		commit
	R(Y)	
	R(Y)	

c)

T1	T2	T3
R(X)		
R(Y)		
W(X)		
		W(Y)
		commit
	R(Y)	
W(X)		
Commit		
	R(Y)	

Bonus Problem 6

So far we have looked at the impact of read and write and how they pertain to a database transaction. Suppose that a new database finds that most of their work is incrementing an decrementing integer records. In this new DBMS they want to introduce a new operations called increment, which increments an integer valued object by 1, and decrement. A transaction that increments an object need not know the value of the object; increment and decrement are version of blind writes. In addition to the shared and exclusive locks, two special locks are supported: an object must be locked in I mode before increment and locked in D mode before decrementing it. An I lock is compatible with another I or D lock on the same object, but not with S and X locks.

1. Illustrate how the use of I and D locks can increase concurrency. (Show a schedule allowed by Strict 2PL that only uses S and X locks. Explain how the use of I and D locks can allow more actions to be interleaved, while continuing to follow Strict 2PL.)

2. Informally explain how Strict 2PL guarantees even in the presence of I and D locks. (Identify which pairs of actions conflict, in the sense that their relative order can affect the result, and show that the us of S, X, I and D locks according toe Strict 2PL orders all conflicting pairs of actions to be the same as the order in some serial schedule.)

Take the following two transactions as example: T1: Increment A, Decrement B, Read C T2: Increment B, Decrement A, Read C
If using only strict 2PL, all actions are versions of blind writes, they have to obtain exclusive locks on objects. Following strict 2PL, T1 gets an exclusive lock on A, if T2 now gets an exclusive lock on B, there will be a deadlock. Even if T1 is fast enough to have grabbed an exclusive lock on B first, T2 will now be blocked until T1 finishes. This has little concurrency. If I and D locks are used, since I and D are compatible, T1 obtains an I-Lock on A, and a D-Lock on B; T2 can still obtain an I-Lock on B, a D-Lock on A; both transactions can be interleaved to allow maximum concurrency.
The pairs of actions which conflicts are: RW, WW, WR, IR, IW, DR, DW We know that strict 2PL orders the first 3 conflicts pairs of actions to be the same as the order in some serial schedule. We can also show that even in the presence of I and D locks, strict 2PL also orders the latter 4 pairs of actions to be the same as the order in some serial schedule. Think of an I (or D)lock under these circumstances as an exclusive lock, since an I(D) lock is not compatible with S and X locks anyway (ie. cant get a S or X lock if another transaction has an I or D lock). So serializability is guaranteed.