University of California, Los Angeles
Department of Computer Science

Computer Science 143 **Prof. Ryan Rosario**

Homework 5

Solutions

Please remember the following:

1. Homework is mostly graded on completion. We may grade a few parts, but it will never be the majority of the grade on the assignment. So try your best, and focus on solving the problems. Consider homework (and studying the solutions) as practice for the final exam.
2. Homework must be submitted digitally, on CCLE. We will not do any paper grading. You can use a text file, but if you use Word, a PDF is preferred rather than a DOC file.
3. Solutions will be posted.
4. **Students are bound by the signed Academic Integrity Agreement. Students copying off of each other (or from other sources) or having unusually similar responses (without citing who they worked with), are easy to identify and will receive a grade of 0.**

1. We will use the following transaction schedule S for this problem. Assume autocommit is enabled.

   This problem was modified from its original form, to make it easier. Unfortunately, the problem became entirely trivial.

T_1	T_2	T_3	T_4
		`write(A)`	
`read(A)`			
`write(B)`			
	`read(B)`		
	`write(C)`		
			`read(B)`

   (a) Is S serial?

   Oh, it's serial! T_1 consists of a single operation and it executes to its completion before the next transaction, T_1 begins to execute. T_1 executes to completion without any interleaving. Then T_2 begins execution and continues to completion without any interleaving. T_4 consists of one operation and it begins executing after all other transactions have completed.

   (b) Is S conflict serializable? If so, what are the equivalent serial schedules?

   Since S is serial, it is conflict serializable by definition.

2. Consider the relation `Googler(name, daysoff)` where we store the number of days off a Googler has remaining this year, and `name` is the key. Suppose we execute the following three transactions.

Thank you for TA Jiaqi for this solution.

T_1:

```
SELECT SUM(daysoff) FROM Googler;
COMMIT;
```

T_2: In this transaction, Google gives everyone an extra day off, and Larry Page gets an additional 10 days off because he is awesome.

```
UPDATE Googler SET daysoff = daysoff + 1;
UPDATE Googler SET daysoff = daysoff + 10 WHERE name = "Larry Page";
```

T_3: We give a few others some more days off in this transaction. We will also set the number of days off for Xooglers to zero.

```
UPDATE Googler SET daysoff = daysoff + 10 WHERE name = "Larry Page";
UPDATE Googler SET daysoff = 0 WHERE name = "James Damore";
```

Unfortunately, there was an error in this problem. This solution will answer the problem as written, then we present the solution for the intended problem. The intended transactions were

T_1:

```
SELECT SUM(daysoff) FROM Googler;
COMMIT;
```

T_2:

```
UPDATE Googler SET daysoff = daysoff + 1;
UPDATE Googler SET daysoff = daysoff + 10 WHERE name = "Larry Page";
COMMIT;    -- this was missing
```

T_3:

```
UPDATE Googler SET daysoff = daysoff + 10 WHERE name = "Larry Page";
UPDATE Googler SET daysoff = 0 WHERE name = "James Damore";
COMMIT;    -- this was missing
```

The `Googler` table originally has two tuples `('Larry Page', 15)` and `('James Damore', 15)`. Assume that *individual* SQL statements execute atomically.

(a) If all three transactions execute under the `SERIALIZABLE` isolation level, list all possible values that can be returned by T_1. Explain your answer.

For the intended problem:

When all transactions run under `SERIALIZABLE`, any possible schedule is conflict equivalent to a serial schedule and is thus conflict serializable. Possible schedules for T1, T2 and T3 are: $\{T_1, T_2, T_3\}$, $\{T_1, T_3, T_2\}$, $\{T_2, T_1, T_3\}$, $\{T_2, T_3, T_1\}$, $\{T_3, T_1, T_2\}$, and $\{T_3, T_2, T_1\}$. The outputs from T_1 are 30, 30, 42, 36, 25, 37 respectively.

(b) If T_1 executes under the `READ UNCOMMITTED` isolation level and T_2 under `REPEATABLE READ` access level, and T_3 under the `SERIALIZABLE` isolation level, list all possible values that can be returned by T_1. Explain your answer.

For the intended problem:

Only T_2 and T_3 are updating values, so let us focus on these two transactions first. Under `REPEATABLE READ`, the only exception to ACID is the phantom phenomenon, but because T_2 and T_3 do not insert any tuple, we do not need to worry about ACID exceptions for the two.

So the possible schedules for T_2 and T_3 are equivalent to $\{T_2, T_3\}$ or $\{T_3, T_2\}$. Regarding T_1, since it is `READ UNCOMMITTED`, its `SELECT` statement may do a dirty read.

Now let us consider the schedule $\{T_2, T_3\}$. Under this schedule, the total daysoff value changes from $30, 32, 42, 52, 36$. T_1 may read any of these values.

For the schedule $\{T_3, T_2\}$, the total daysoff value changes from $30 \rightarrow 40 \rightarrow 25 \rightarrow 27 \rightarrow 37$. Again, T_1 may read any of these sum values.

Solution for problem 2ab, as written:

In this situation, each statement in T_2 or T_3 is treated as `autocommit`. The isolation level does not make difference in this case as each statement in T_2 or T_3 is executed as in its own transaction. Both (a) and (b) will produce the same answer. A possible schedule is a permutation of T1 T2-T1 T2-2 T3-1 T3-2 (we also need to make sure in any valid schedule, T2-1 is always before T2-2 and the same for T3-1/T3-2), thus a possible answer is the sum of `daysoff` values that T_1 reads (on the moment that T_1 executes) in that possible schedule.

Larry Page is the CEO of Alphabet, the parent company of Google. He also served as the CEO before Google reorganized under Alphabet. PageRank, an algorithm for measuring how "influential" a web page is based on its in/out links, was named after him (not for the fact it considers web pages). James Damore on the other hand, was the most recent high profile firing from Google after he published a sexist manifesto that rocked the tech world.