Assignment: Databases—Inside the Blackbox

COMPSCI 2DB3: Databases–Winter 2024

Deadline: April 10, 2023*

Department of Computing and Software
McMaster University

Please read the *Course Outline* for the general policies related to assignments.

**Plagiarism is a *serious academic offense* and will be handled accordingly.
All suspicions will be reported to the *Office of Academic Integrity*
(in accordance with the Academic Integrity Policy).**

This assignment is an *individual* assignment: do not submit work of others. All parts of your submission *must* be your own work and be based on your own ideas and conclusions. Only *discuss or share* any parts of your submissions with your TA or instructor. You are *responsible for protecting* your work: you are strongly advised to password-protect and lock your electronic devices (e.g., laptop) and to not share your logins with partners or friends!

If you *submit* work, then you are certifying that you are aware of the *Plagiarism and Academic Dishonesty* policy of this course outlined in this section, that you are aware of the Academic Integrity Policy, and that you have completed the submitted work entirely yourself. Furthermore, by submitting work, you agree to automated and manual plagiarism checking of all submitted work.

Late submission policy. Late submissions will receive a late penalty of 20% on the score per day late (with a five hour grace period on the first day, e.g., to deal with technical issues) and submissions five days (or more) past the due date are not accepted. In case of technical issues while submitting, contact the instructor *before* the deadline.

Description

Recently, a game introduced an in-game crafting and trading system that has seen explosive usage among the players. Due to this, the online components of the game have seen a decrease in performance. Hence, the game developers asked for a consultant to design a higher-performance system for crafting and trading. The consultant took a look at the original system and concluded that *lock-based* access to player inventories was the main culprit. To deal with these performance issues, the consultant came up with a *novel* design that reduces the duration of locks. Next, we detail the proposed design.

In the system, all crafting and trading operations can be written as a sequence of *transfers* of the form

$$\underline{\text{transfer}}(\$x, \mathit{from}, \mathit{to}) = \text{“transfer object \$x from the inventory of player } \mathit{from} \text{ to the inventory of player } \mathit{to}\text{”},$$

that should only be executed if *each transfer* is possible (the inventory of the *from*-player has one-or-more copies of the transferred object). E.g., the transaction

$$\tau = [\underline{\text{transfer}}(\mathit{Apple}, \mathit{Bo}, \mathit{Alicia}), \underline{\text{transfer}}(\mathit{Pear}, \mathit{Eva}, \mathit{Celeste})]$$

*This assignment is intended as a *bonus* and as an opportunity to improve ones grades if needed. Remember that the best seven out of eight assignments will count toward your final grade. If you are happy with your previous assignments, then you are *not* required to make this assignment. The material covered in this assignment will *not* be part of the final exam.

```
UPDATE-INVENTORY(τ, account, object, remove):
1: Lock_τ(account).
2: if remove = True then
3:    Remove one copy of object from the inventory of account.
4: else
5:    Add one copy of object to the inventory of account.
6: end if
7: Release_τ(account).
```

Figure 1: The pseudo-code for the minimal-locking operation UPDATE-INVENTORY that can add or remove a single object from the inventory of an account.

```
EXECUTE-TRANSACTION(τ):
1: Commit, Rollback := ∅, ∅.
2: for each transfer($x, from, to) in τ do
3:    Lock_τ(from).
4:    if from holds a copy of $x then
5:       Release_τ(from).
6:       Store the operation "UPDATE-INVENTORY(τ, from, $x, True)" in Commit.
7:       Store the operation "UPDATE-INVENTORY(τ, to, $x, False)" in Commit.
8:    else
9:       Release_τ(from).
10:      return failure.
11:   end if
12: end for
13: Perform all operations in Commit.
14: return success.
```

Figure 2: The pseudo-code for the transaction execution algorithm.

of two such transfers is equivalent to

> "if *Bo* has an Apple and *Eva* has a Pear,
> then transfer the Apple from *Bo* to *Alicia* and transfer the Pear from *Eva* to *Celeste*".

To model crafting of a *new* object, the system supports transfers from a special-purpose *system*-inventory that always has sufficient copies of all objects.

The consultant wants to execute these transactions with a minimal amount of locking. To do so, the consultant designed the minimal-locking operation UPDATE-INVENTORY (see Figure 1 for the pseudo-code of this operation). Using this operation, the consultant proposes to execute transactions τ with n *transfers*, e.g., $\tau = [\underline{\text{transfer}}(\$x_1, from_1, to_1), \ldots, \underline{\text{transfer}}(\$x_n, from_n, to_n)]$, using the EXECUTE-TRANSACTION algorithm (see Figure 2 for the pseudo-code of this algorithm). The EXECUTE-TRANSACTION algorithm will visit each *from*-player inventory and check whether the inventory of that player has the to-be transferred object. Only if these checks hold for all objects $\$x_1, \ldots \$x_n$, then the algorithm will transfer all these objects to the inventory of the respective $to_1, \ldots, to_n$ players.

To do so, the variable *Commit* lists all UPDATE-INVENTORY operations necessary to move all objects at once after we have checked whether each individual inventory of a *from*-player has sufficient objects. If the inventory of a *from*-player is found without objects, then no objects will be transferred.

The consultant believes that this setup will reduce the duration in which individual locks are held,

but might introduce unwanted interference between transactions. The game developer has already been instructed about the risks of interference, and decided that it can agree to interference as long as the following *constraints* are never broken:

C1. No inventory should ever receive a negative number of objects (assuming that all inventories start with a positive number of objects).

C2. As the transfers only move objects between inventories, no objects should be *lost* or *created*. Hence, if at any time t no transactions are being executed, then the bag of objects in all inventories at that time t should be equivalent to the initial bag of objects of all inventories.

C3. Successful transactions must have their *lasting effects*, while failed transactions must not have lasting effects. Hence, if at any time t no transactions are being executed, then the objects in each inventory should reflect the updates due to all transactions that executed successfully before t.

We note that these constraints do not rule out *inconsistencies* in the data while transactions are being executed.

Faced with the complexity of the approach proposed by the consultant, the game developer has contacted you to evaluate the proposed approach.

Your evaluation

To evaluate the approach, the game developer asked you to investigate and answer the following questions:

1. Does the proposed approach follow strict two-phase locking? Does the proposed approach follow two-phase locking? Explain your answer. E.g., if the approach does not follow (strict) two-phase locking, then provide a transaction, its execution schedule, and argue that this schedule does not follow the (strict) two-phase locking protocol.
2. Does the proposed approach suffer from *deadlocks*? Explain your answer.
3. Does the proposed approach expose *read-write*, *write-read*, or *write-write* conflicts? Explain your answer. E.g., if there are conflicts, then provide two transactions, a valid interleaved execution schedule for these transactions, and argue that this schedule has these conflicts.
4. Does the proposed approach suffer from *dirty reads*? Explain your answer. E.g., if there are dirty reads, then provide two transactions, a valid interleaved execution schedule for these transactions, and argue that this schedule has dirty reads.
5. Does the proposed approach suffer from *unrepeatable reads*? Explain your answer. E.g., if there are unrepeatable reads, then provide two transactions, a valid interleaved execution schedule for these transactions, and argue that this schedule has unrepeatable reads.
6. Is the proposed approach *serializable*? Explain your answer.
7. Are the constraints C1-C3, as set out by the game developer, satisfied? Explain your answer. E.g., if a constraint is satisfied, then argue why that is the case.

Assignment

The goal of the assignment is to help out the game developer. To do so, you will write a report in which you answer Questions 1–7. Your submission:

1. must be a `PDF` file;
2. must have clearly labeled solutions to each of the stated evaluation questions;

3. must include explanations for each of the stated evaluation questions;
4. must be clearly presented;
5. must *not* be hand-written: prepare your document in Microsoft Word or another word processor (printed or exported to PDF) or in LaTeX.

Submissions that do not follow the above requirements will get a grade of zero.

Grading

The presented solution for Question 1–6 will each account for 12.5% of the maximum grade; the presented solution for Question 7 will account for 25% of the maximum grade.