Example Assignment:
Databases—Inside the Black Box
COMPSCI 2DB3: Databases

Jelle Hellings Holly Koponen

Department of Computing and Software
McMaster University

Description

A financial service provider came to the conclusion that their system, which processes complex money transfers between people, is not fast enough. Hence, they asked a consultant to come up with a higher-performance system. The consultant took a look at the original system and concluded that *lock-based* access 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 transactions can be written as a sequence of *transfers* of the form

$$\underline{\text{transfer}}(\$x, \mathit{from}, \mathit{to}) = \text{“transfer \$x from account } \mathit{from} \text{ to account } \mathit{to}\text{”},$$

that should only be executed if *each transfer* is possible (the *from*-account has sufficient funds). E.g., the transaction

$$\tau = [\underline{\text{transfer}}(\$500, \mathit{Bo}, \mathit{Alicia}), \underline{\text{transfer}}(\$300, \mathit{Eva}, \mathit{Celeste})]$$

of two such transfers is equivalent to

> “if *Bo* has at-least \$500 and *Eva* has at-least \$300,
> then transfer \$500 from *Bo* to *Alicia* and transfer \$300 from *Eva* to *Celeste*”.

The consultant wants to execute these transactions with a minimal amount of locking. To do so, the consultant designed the minimal-locking operations UPDATE-BALANCE and TAKE-BALANCE-CONDITIONAL (see Figure 1 for the pseudo-code of these operations). Using these operations, the consultant proposes to execute transactions τ with n *transfers*, e.g., $\tau = [\underline{\text{transfer}}(\$x_1, \mathit{from}_1, \mathit{to}_1), \ldots, \underline{\text{transfer}}(\$x_n, \mathit{from}_n, \mathit{to}_n)]$, using the EXECUTE-TRANSACTION algorithm (see Figure 2 for the pseudo-code of this algorithm). The EXECUTE-TRANSACTION algorithm will visit each *from*-account, check whether that account has sufficient funds (at-least the funds required for the transfer), and take away the funds that are to-be transferred (a *reservation of funds*). This reservation can be used in two ways:

1. If all *from*-accounts have sufficient funds, then all reserved funds will be transferred to their respective *to*-accounts (the transaction is successful). To do so, the variable *Commit* lists all UPDATE-BALANCE operations necessary to transfer reserved funds to their respective *to*-accounts.

2. Otherwise, if a *from*-account is found without sufficient funds, then all previously reserved funds will be returned to their respective *from*-accounts (the transaction failed). To do so, the variable *Rollback* lists all UPDATE-BALANCE operations necessary to transfer reserved funds back to their respective *from*-accounts.

```
    UPDATE-BALANCE(τ, account, amount):
 1: Lock_τ(account).
 2: account := account + amount.
 3: Release_τ(account).

    TAKE-BALANCE-CONDITIONAL(τ, account, amount):
 4: Lock_τ(account).
 5: if account ≥ amount then
 6:     account := account − amount.
 7:     Release_τ(account).
 8:     return True.
 9: else
10:     Release_τ(account).
11:     return False.
12: end if
```

Figure 1: The pseudo-code for the minimal-locking operations UPDATE-BALANCE and TAKE-BALANCE-CONDITIONAL.

```
    EXECUTE-TRANSACTION(τ):
 1: Commit, Rollback := ∅, ∅.
 2: for each transfer($x, from, to) in τ do
 3:     if TAKE-BALANCE-CONDITIONAL(τ, from, $x) then
 4:         Store the operation "UPDATE-BALANCE(τ, to, $x)" in Commit.
 5:         Store the operation "UPDATE-BALANCE(τ, from, $x)" in Rollback.
 6:     else
 7:         Perform all operations in Rollback.
 8:         return failure.
 9:     end if
10: end for
11: Perform all operations in Commit.
12: return success.
```

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

The consultant believes that this setup will reduce locking, but might introduce unwanted interference between transactions. The financial service provider 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 account should ever receive a negative balance (assuming that all accounts start with a positive balance).

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

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 balance of each account should reflect the balance 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 financial service provider has contacted you to evaluate the proposed approach.

Your evaluation

To evaluate the approach, the financial service provider 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 financial service provider, 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 financial service provider. To do so, you have to answer Questions 1–7.