Banking System Codewriting Instructions

Introduction

Your task is to implement a simplified version of a banking system. This system must support various operations as detailed below. The solution will be divided into multiple levels. Each level builds upon the functionality of the previous one, and subsequent levels will be opened after successfully solving the current one.

You can execute a single test task by running the following command in the terminal:

```
bank_run_single_test task_id=<task_id>
```

Notes

Timestamps: All operations will include a timestamp parameter — a simplified timestamp in milliseconds. It is guaranteed that all timestamps are unique and are in the range of 1 to 10^12. Operations will be given in the order of strictly increasing timestamps.

Requirements

The banking system must be designed according to the specifications of the levels below:

Level 1: Basic Banking Operations

The banking system should support creating new accounts, depositing money into accounts, and transferring money between two accounts.

Supported Operations

1. createAccount(int timestamp, String accountId): boolean
  - Creates a new account with the given identifier if it doesn't already exist.
  - Returns: True if the account was successfully created, False if an account with the same accountId already exists.

2. deposit(int timestamp, String accountId, int amount): int
  - Deposits the specified amount of money into the given account.
  - Returns: The balance of the account after the deposit has been processed.
  - Fails: return -1 If the account does not exist.

3. transfer(int timestamp, String sourceAccountId, String targetAccountId, int amount): int
  - Transfers the given amount of money from the source account to the target account.
  - Returns: The balance of the source account after the transfer is successful or -1 if the

transfer cannot be made.
  - Fails: return -1 If either the source account or the target account doesn't exist, the two accounts are the same, or if there are insufficient funds in the source account.

Level 2: Ranking Accounts Based on Outgoing Transactions

The bank wants to identify people who are not keeping money in their accounts, so implement operations to support ranking accounts based on outgoing transactions.

Supported Operations

1. topSpenders(int timestamp, int n): List<String>

- Should return the list of top-n accounts with the highest outgoing transactions - the total amount of money either transferred out or paid/withdrawn (the Pay operation will be introduced in level 3)
- Sorting: in descending order or in case of tie, sorted alphabetically by account_id in ascending order.
- The result should be a list of strings in the following format: ['account_id_1:total_outgoing_1', 'account_id_2:total_outgoing_2', ...]
- If less than n accounts exist in the system, then return all their identifiers (in the described format).
- Cashback (an operation that will be introduced in level 3) should not be reflected in the calculations for total outgoing transactions.

Level 3: Scheduling Payments with Cashback

The banking system should allow scheduling payments with cashback and checking the status of scheduled payments.

Supported Operations

1. pay(int timestamp, String accountId, int amount): String | None

Should withdraw the given amount of money from the specified account. All withdrawal transactions provide a 2% cashback - 2% of the withdrawn amount (rounded down to the nearest integer) will be refunded to the account 24 hours after the withdrawal. if the withdrawal is successful. (i.,e; the account holds sufficient funds to withdraw the given amount). returns a string with a unique identifier for the payment transaction in this format.

"payment(ordinal number of withdrawals from all accounts)". e.g; "payment1", "payment2", etc. Additional conditions:

- Returns None if **accountId** does not exist.
- Returns None if **accountId** has insufficient funds to perform the payment.
- **top_spenders** should now also account for the total amount of money withdrawn from accounts.
- The waiting period for cashback is 24 hours, equal to 86400000 milliseconds(the unit for timestamp). So, cashback will be processed at timestamp (timestamp+86400000).

- When it's time to process cashback for a withdrawal, the amount must be refunded to the account before any other transactions are performed at the relevant timestamp.

2. getPaymentStatus(int timestamp, String accountId, String payment): String | None
 - Returns the status of the payment transaction for the given payment: "**IN_PROGRESS**" or "**CASHBACK_RECEIVED**".
 - Fails:
  - If the accountId does not exist.
  - If the payment does not exist for the specified account.

Level 4: Merging Accounts

The banking system should support merging two accounts while retaining both accounts' balance and transaction histories.

Supported Operations

1. mergeAccounts(int timestamp, String accountId1, String accountId2): boolean
 - Merges accountId2 into accountId1.
 - Returns: True if accounts were successfully merged, False otherwise.

- Return false If accountId1 is equal to accountId2
- Return false If either accountId1 or accountId2 does not exist.
- All pending cash back refunds to accountId2 should still be processed, but refunded to accountId1.
- After the merge, it must be possible to check the status of payment transactions for accountId2 *with payment identifiers by replacing* accountId2 with accountId1.
- **top_spenders** operations should recognize merged accounts - the total outgoing transactions for merged accounts should be the sum of all money transferred and/or withdrawn in both accounts.
- accountId2 should be removed from the system after merge.

2. getBalance(int timestamp, String accountId, int timeAt): int | boolean
 Should return the total amount of money in the accountId at the given timestamp **timeAt**. *If the specified account did not exist at a given time, timeAt*, return false.

- If queries have been processed at timestamp timeAt, getBalance must reflect the account balance after the query has been processed.
- if the account was merged into another account, the merged account should inherit the balance history.