Rust Coding Test

Thanks for taking the time to try out our coding challenge. Since we don't do a lot of whiteboarding during the interview process we've found take home problems to be the best way for you to show us your skills. This gives you the chance to think about the problem, solve it when you feel comfortable, and focus on the areas you think are important. This challenge should take you 2-3 hours on the keyboard, but we encourage you to read the problem description and take some time to think about your solution before you dive into it :)

Overview

At Kraken we deal with transactions on and off chain. Off chain transactions are generally mutable, and in a lot of ways this a good thing. It lowers the risk of losing your credit card because you know your credit provider has your back, but for vendors integrating into payment networks or with payment partners it means we need to be vigilant and on the lookout for fraudsters. For example, a malicious actor may try to deposit fiat funds, purchase and withdraw BTC, and then reverse their fiat deposit.

We'd like you to implement a simple toy payments engine that reads a series of transactions from a CSV, updates client accounts, handles disputes and chargebacks, and then outputs the state of clients accounts as a CSV.

Scoring

You will be scored on the following criteria

Category	Description
Basics	Does your application build? Does it read and write data in the way we'd like it to? Is it properly formatted?
Completeness	Do you handle all of the cases, including disputes, resolutions, and chargebacks? Maybe you don't handle disputes and resolutions but you can tell when a transaction is charged back. Try to cover as much as you can
Correctness	For the cases you are handling, are you handling them correctly? How do you know this? Did you test against sample data? If so, include it in the repo. Did you write unit tests for the complicated bits? Or are you using the type system to ensure correctness? Tell us about it in the README

Safety and Robustness	Are you doing something dangerous? Tell us why you chose to do it this way. How are you handling errors?
Efficiency	Be thoughtful about how you use system resources. Sample data sets may be small but you may be evaluated against much larger data sets (hint: transaction IDs are valid u32 values). Can you stream values through memory as opposed to loading the entire data set upfront? What if your code was bundled in a server, and these CSVs came from thousands of concurrent TCP streams?
Maintainability	In this case clean code is more important than efficient code because humans will have to read and review your code without an opportunity for you to explain it. Inefficient code can often be improved if it is correct and highly maintainable.

Your solution will be scored using a combination of automated and manual scoring. Automated scoring will be used to run your solution against a handful of sample inputs, comparing the resulting output. If our automatic test suite passes, manual scoring will be used for everything else. At Kraken we care about clean, correct, safe, and efficient code.

Because we use some automated scoring, **it is very important your program exposes the CLI interface described below**. Anything else will be ignored by reviewers. If you think there are inconsistencies in the specification, the described input & output files take precedence.

Details

Given a CSV representing a series of transactions, implement a simple toy transactions engine that processes the payments crediting and debiting accounts. After processing the complete set of payments output the client account balances

You should be able to run your payments engine like

```
$ cargo run -- transactions.csv > accounts.csv
```

The input file is the first and only argument to the binary. Output should be written to std out

Input

The input will be a CSV file with the columns type, client, tx, and amount. You can assume the type is a string, the client column is a valid u16 client ID, the tx is a valid u32 transaction ID, and the amount is a decimal value with a precision of up to four places past the decimal.

For example

```
type, client, tx, amount
deposit, 1, 1, 1.0
deposit, 2, 2, 2.0
deposit, 1, 3, 2.0
withdrawal, 1, 4, 1.5
withdrawal, 2, 5, 3.0
```

The client ID will be unique per client though are not guaranteed to be ordered. Transactions to the client account 2 could occur before transactions to the client account 1. Likewise, transaction IDs (tx) are globally unique, though are also not guaranteed to be ordered. You can assume the transactions occur chronologically in the file, so if transaction b appears after a in the input file then you can assume b occurred chronologically after a. Whitespaces and decimal precisions (up to four places past the decimal) must be accepted by your program.

Output

The output should be a list of client IDs (client), available amounts (available), held amounts (held), total amounts (total), and whether the account is locked (locked). Columns are defined as

Column	Description
available	The total funds that are available for trading, staking, withdrawal, etc. This should be equal to the total - held amounts
held	The total funds that are held for dispute. This should be equal to total - available amounts
total	The total funds that are available or held. This should be equal to available + held
locked	Whether the account is locked. An account is locked if a charge back occurs

For example

```
client, available, held, total, locked
     1, 1.5, 0.0, 1.5, false
     2, 2.0, 0.0, 2.0, false
```

Spacing and displaying decimals for round values do not matter. Row ordering also does not matter. The above output will be considered the exact same as the following

```
client,available,held,total,locked
2,2,0,2,false
1,1.5,0,1.5,false
```

Precision

You can assume a precision of **four places past the decimal** and should output values with the same level of precision.

Types of Transactions

Deposit

A deposit is a credit to the client's asset account, meaning it should increase the available and total funds of the client account

A deposit looks like

type	client	tx	amount
deposit	1	1	1.0

Withdrawal

A withdraw is a debit to the client's asset account, meaning it should decrease the available and total funds of the client account

A withdrawal looks like

type	client	tx	amount
withdrawal	2	2	1.0

If a client does not have sufficient available funds the withdrawal should fail and the total amount of funds should not change

Dispute

A dispute represents a client's claim that a transaction was erroneous and should be reversed.

The transaction shouldn't be reversed yet but the associated funds should be held. This means that the clients available funds should decrease by the amount disputed, their held funds should increase by the amount disputed, while their total funds should remain the same.

A dispute looks like

type	client	tx	amount
dispute	1	1	

Notice that a dispute does not state the amount disputed. Instead a dispute references the transaction that is disputed by ID. If the tx specified by the dispute doesn't exist you can ignore it and assume this is an error on our partners side.

Resolve

A resolve represents a resolution to a dispute, releasing the associated held funds. Funds that were previously disputed are no longer disputed. This means that the clients held funds should decrease by the amount no longer disputed, their available funds should increase by the amount no longer disputed, and their total funds should remain the same.

A resolve looks like

type	client	tx	amount
resolve	1	1	

Like disputes, resolves do not specify an amount. Instead they refer to a transaction that was under dispute by ID. If the tx specified doesn't exist, or the tx isn't under dispute, you can ignore the resolve and assume this is an error on our partner's side.

Chargeback

A chargeback is the final state of a dispute and represents the client reversing a transaction. Funds that were held have now been withdrawn. This means that the clients held funds and total funds should decrease by the amount previously disputed. If a chargeback occurs the client's account should be immediately frozen.

A chargeback looks like

type	client	tx	amount
chargeback	1	1	

Like a dispute and a resolve a chargeback refers to the transaction by ID (tx) and does not specify an amount. Like a resolve, if the tx specified doesn't exist, or the tx isn't under dispute, you can ignore chargeback and assume this is an error on our partner's side.

Submission

When you are ready to share your submission, please email us back and include a link to a **GitHub** repository containing your solution. The repository should be a simple Rust crate generated using cargo new (or cargo init), should be buildable via cargo build, and runnable via cargo run.
Your solution should not mention Kraken, Payward, or any associated Kraken products, brands, web domains, etc. This test file or any derivative must not be committed.

Assumptions

You're safe to make the following assumptions

- The client has a single asset account. All transactions are to and from this single asset account;
- There are multiple clients. Transactions reference clients. If a client doesn't exist create a new record;
- Clients are represented by `u16` integers. No names, addresses, or complex client profile info;

When in doubt on how to interpret a requirement, try to make assumptions that make sense for a bank (think an ATM or more elaborate transaction processors), and document them.

Useful Libraries!

- Serde for serialization and deserialization
- csv for reading and writing CSVs
- Any other common crate that you deem secure.