Online Retail Store

Scope:

With the advent of the Covid-19 pandemic, the popularity of online applications, in general, has skyrocketed. One such use case which has seen tremendous growth in terms of usability is that of e-commerce websites. It allows users to shop from the comfort of their homes while eliminating the need to worry about long queues, inflexible working hours, and having to navigate/ search for a product in offline retail stores.

In this project, we aim to build our own online retail management system. It offers different functionalities to clients and store managers.

Client-side features include the option to have a personalised account which can then be used to manage their orders.The management side allows the store to effectively manage various aspects of finances, such as maintaining accounts of transactions, recording profit/losses, and keeping track of the products currently in the inventory. This part of the application will only be accessible to the admin.

Functional Requirements:

The online retail store provides the following functions to clients/users:

- Allows customers to create an account(Sign in with authentication), store their details, and use those to handle discounts and other offers.
- Supports an in-store shopping cart that dynamically updates and cross-checks with the current inventory. It updates the inventory when products are added/removed from the cart and generates the bill on checkout.
- Allows users to filter products based on type, price, etc., to streamline the entire process.

Additional features offered on the store management side are:

- Keeping an inventory for the department store and tracking different categories and products that are available for sale in the store.
- Handling finances relating to the purchase and sale of different goods, generating profit and loss statements at different intervals.

Stakeholders:

- **Customers/ Users** looking to purchase goods conveniently.
- **Store Operators** looking to maximise efficiency in managing store accounts and product inventory.

Technical Requirements:

Data:

- The store's internal records are protected via authentication upon login and cannot be accessed by other users. Similarly a customer can only access their own cart.
- The various filters and dynamic shopping cart allows users to search for and keep track of products they wish to purchase more conveniently
- Ensures no concurrency issues by ensure each addition/removal from the cart is not duplicated between users thus preventing inventory tracking issues

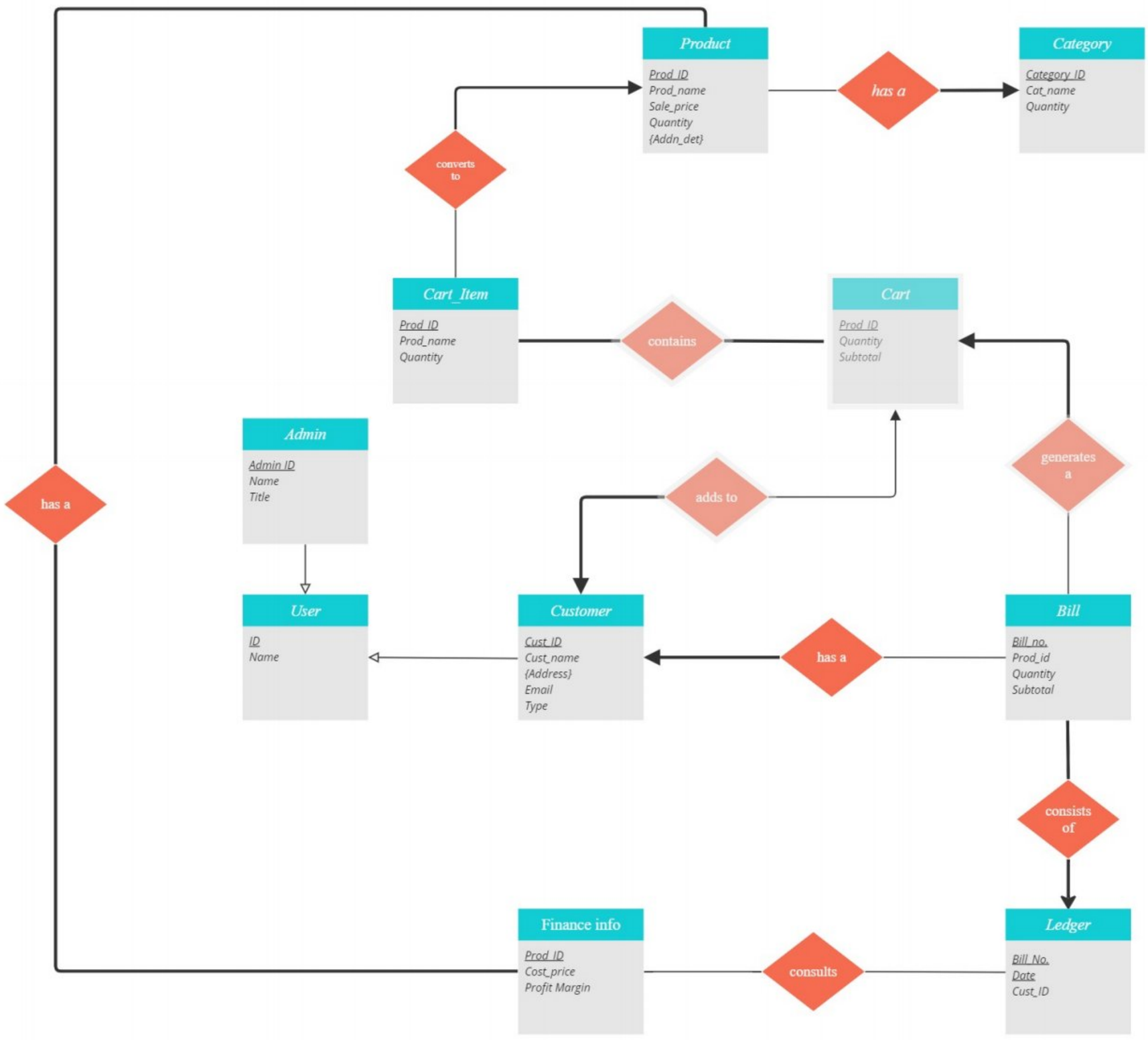
Backend:

- MySQL
- Python
- Django

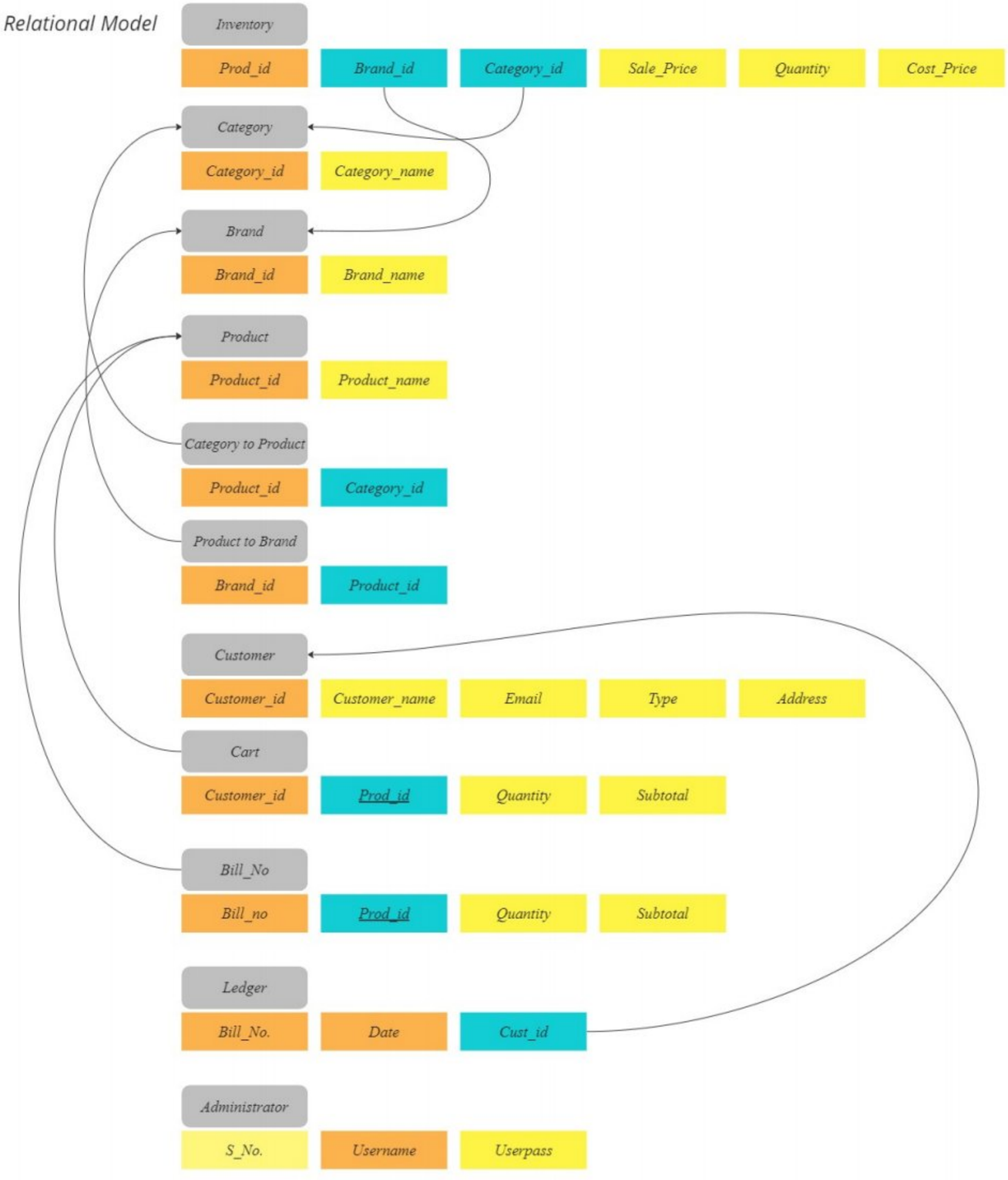
Frontend:

- HTML
- CSS
- JavaScript
- Bootstrap

ER Diagram



Updated Relational Model



FUNCTIONS AND SQL QUERIES

Customer side:

- Let a customer see their purchase history for a particular time period.
  (Eg: Find Prod_name, Qty, Subtotal for all products purchased by Lisa in March 2020)
  QUERY- 1

- Let a customer browse a kind of product under a certain price.
  (Eg: Find Brand_name, Price, Qty for all brands of Milk and under Rs.55)
  QUERY- 2

- Let a customer see all products of a specific category that they purchased from most costly to least
  (Eg: Find Brand_name, Prod_name, Price for all products purchased by Lisa under the category of Food)
  QUERY- 3

- Let customers repeat an order from their past history, and the customer will provide bill no.
  QUERY- 8

Admin Side:

- Let the admin see which products of each category are left in the inventory that have a qty of less than a (predetermined) value
  (Eg: Find Category_name, Prod_name, quantity for all products whose quantity is less than 5, grouped by category)
  QUERY- 4

- Let the admin see the top 10 customers who have the most bills in his store in ordered by the number of their bills
  (Eg: Find Cust_name, No_of_Bills of those 10 customers that have most orders in this store)
  QUERY- 5

- Let admin see all the categories of his store in order of most profitable to least profitable (Eg: Find Cat_name, Profit_Per for all categories ordered from highest profit to least profit)
  QUERY- 6

- Let admin see profit of given time period
  (Eg: Find profit percentage of store made in month of March)
  QUERY- 7

- Finding the most luxurious (Case 1) or cost-effective (Case 2) brand in the store looking at the average price of the products sold by it
  QUERY- 9

- Admin updates the type of a customer
  (Eg: Customer type is upgraded after customer_id = 2 makes payment)
  QUERY- 10

- Update the customer table to include a status that tells us whether the customer is active or not
  QUERY- 11

OLAP QUERIES

1. How many non distinct products were sold in each category each year ?

2. How has been the brand-wise profit (in Rs.) each year been like ?

3. Describe the total amount spent by a customer (in Rs.) in each month,year for buying stuff ?

4. What is the profit percentage for each brand of every product (to decide which brand is beneficial to buy products to stock the store) ?

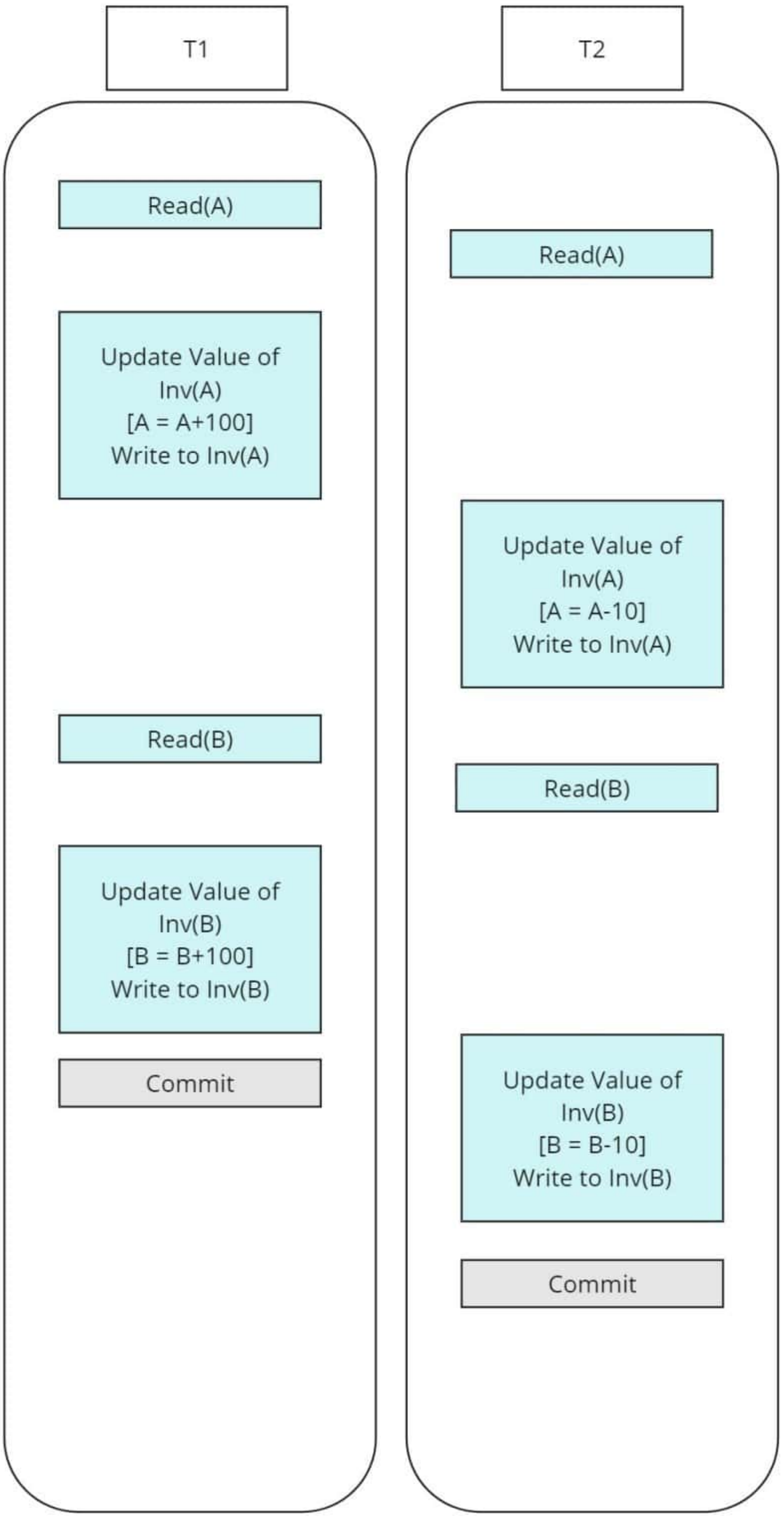
TRIGGERS

1. For generating final bill for a person while chekout.

2. For preventing invalid customer types: like negatives and zeros.

3. For preventing invalid customer names : like those having numbers in names

Transactions

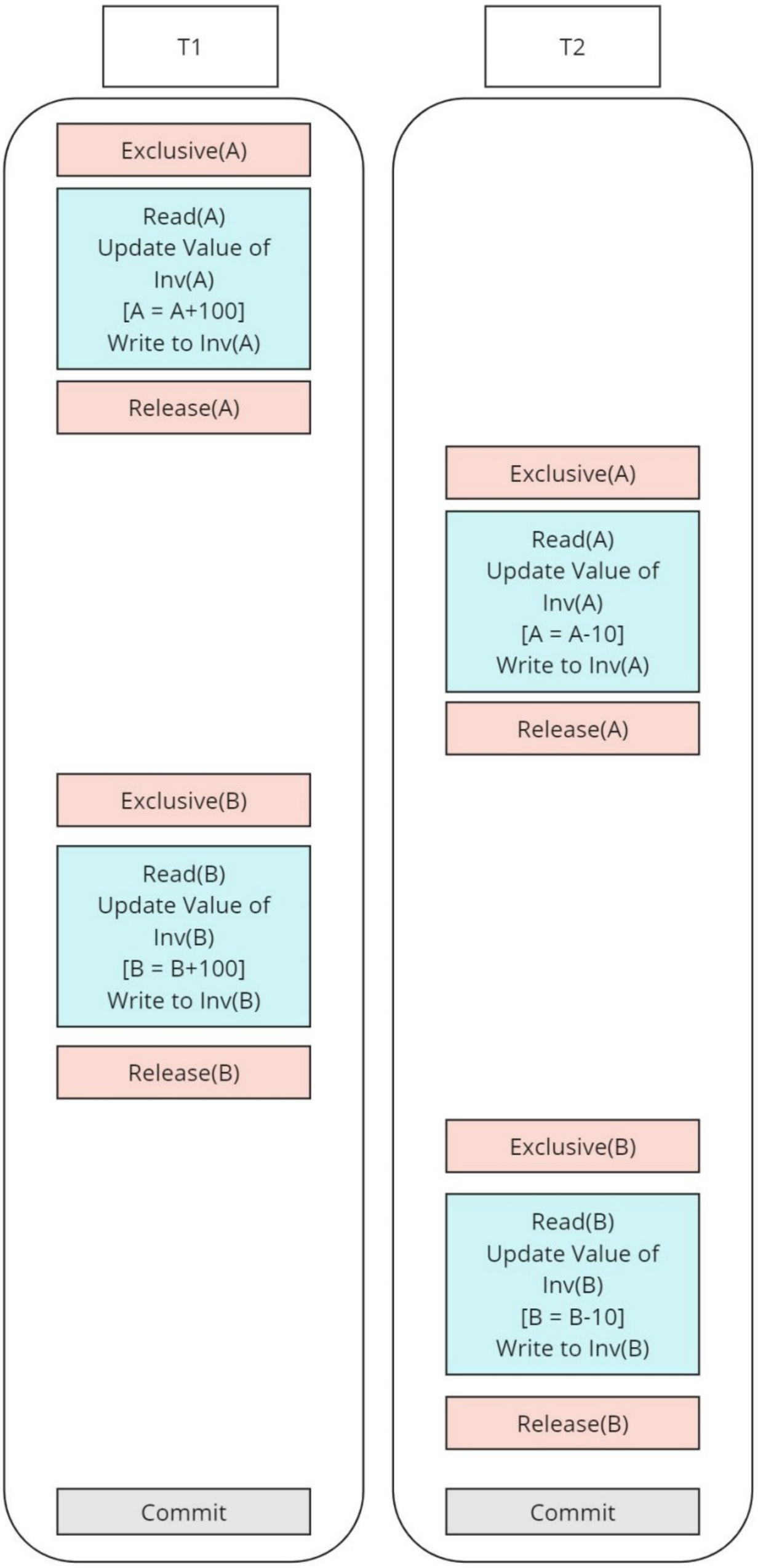


Problem Description
NewT1

A customer tries to place an order for items A & B (quantity in cart is 10), simultaneously admin attempts to add 100 to the quantity of items A & B in inventory.

Here the initial quantity of A & B in inventory is **more than 10** thus admin's addition doesn't affect the transaction.

Non-Conflict Serializable



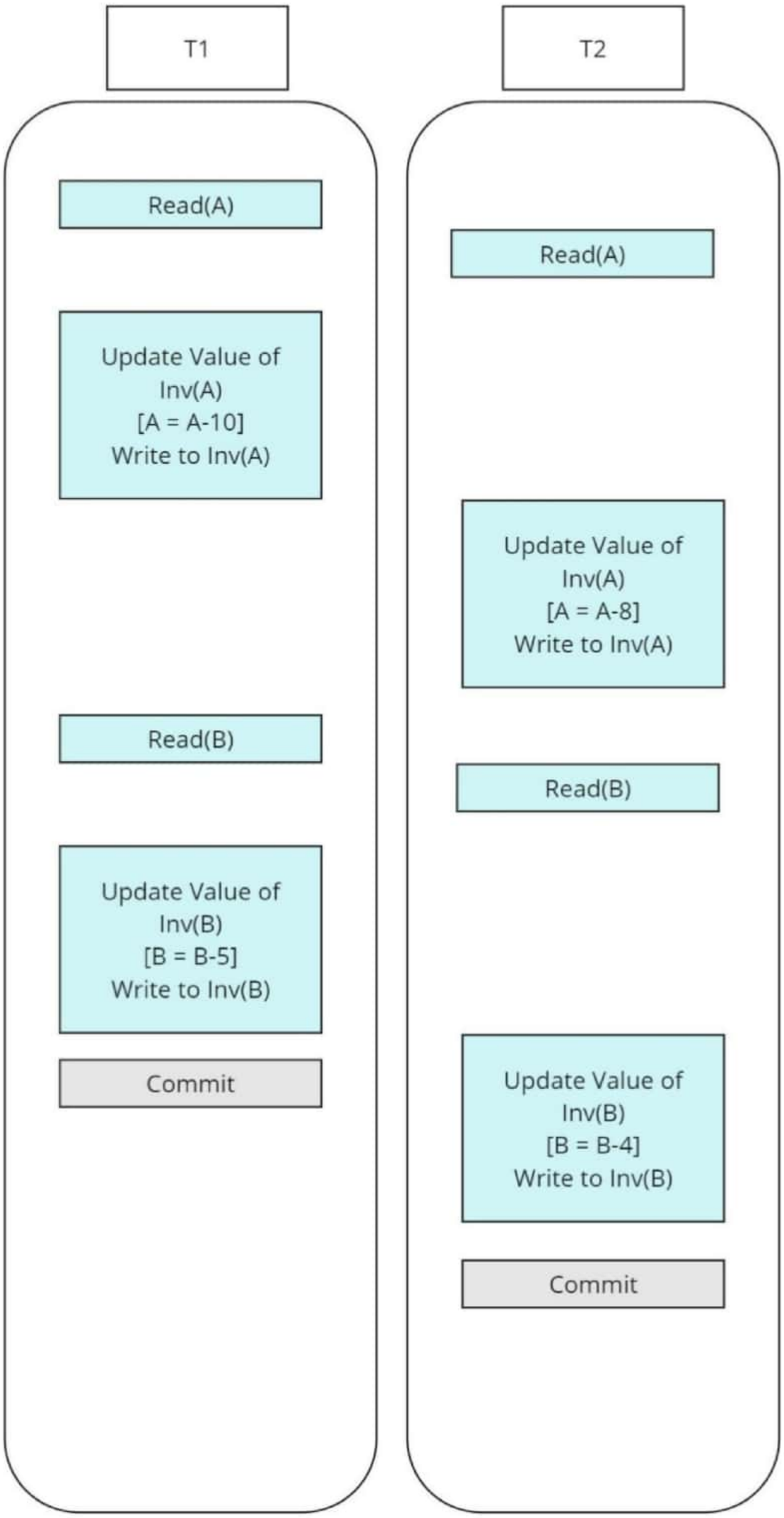
Problem Description
NewT1

A customer tries to place an order for items A & B (quantity in cart is 10), simultaneously admin attempts to add 100 to the quantity of items A & B in inventory.

Here the initial quantity of A & B in inventory is **more than 10** thus admin's addition doesn't affect the transaction.

Conflict Serializable

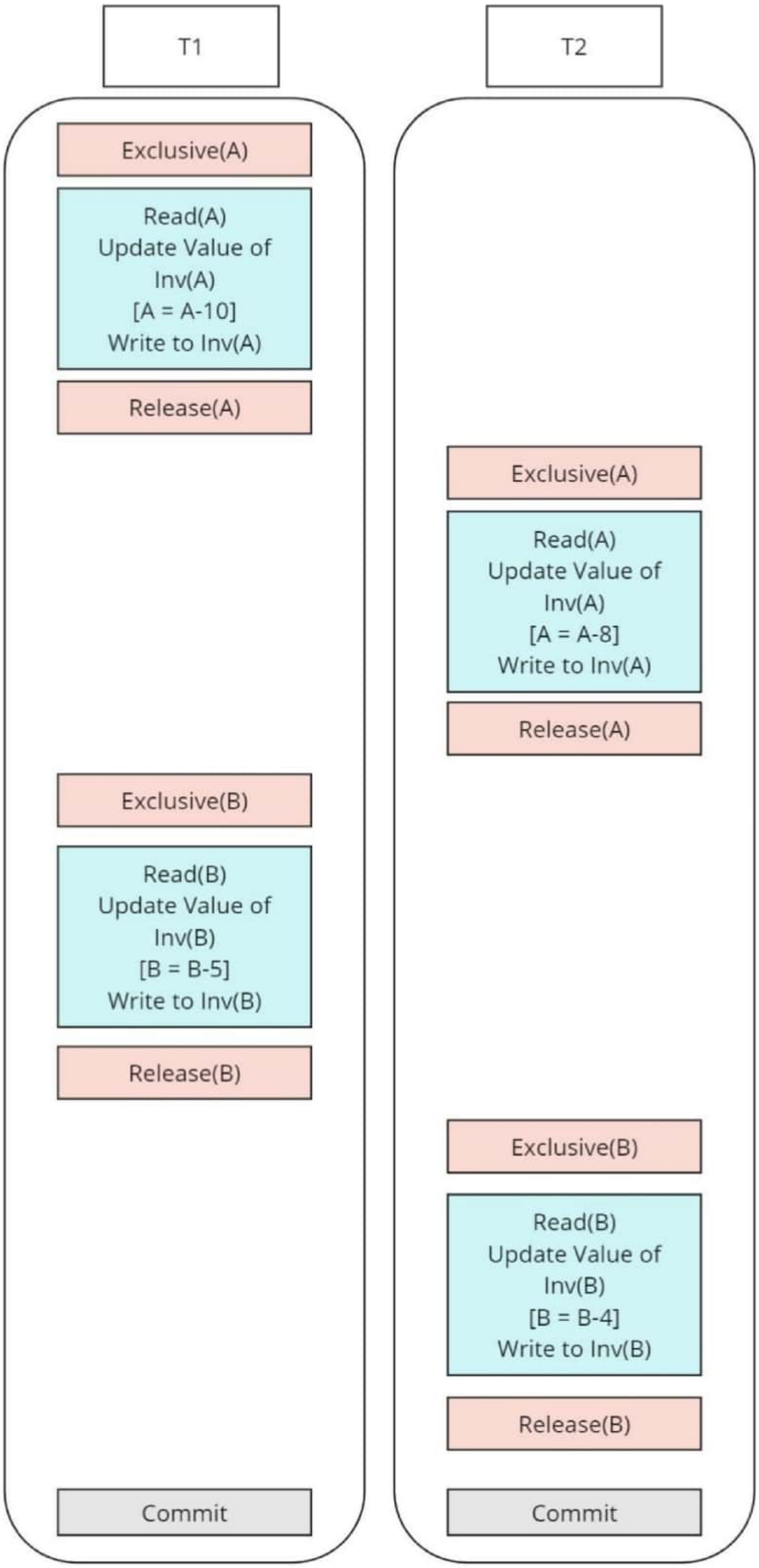
Problem Description
New T2



2 Customers try to place orders for items A & B (quantity in cart is 10,5 & 8,4), simultaneously

Here the initial quantity of A & B in inventory is **more than** the sum of quantities being ordered by each customer so the order placement times don't matter

Non-Conflict Serializable



Problem Description

New T2

2 Customers try to place orders for items A & B (quantity in cart is 10,5 & 8,4), simultaneously

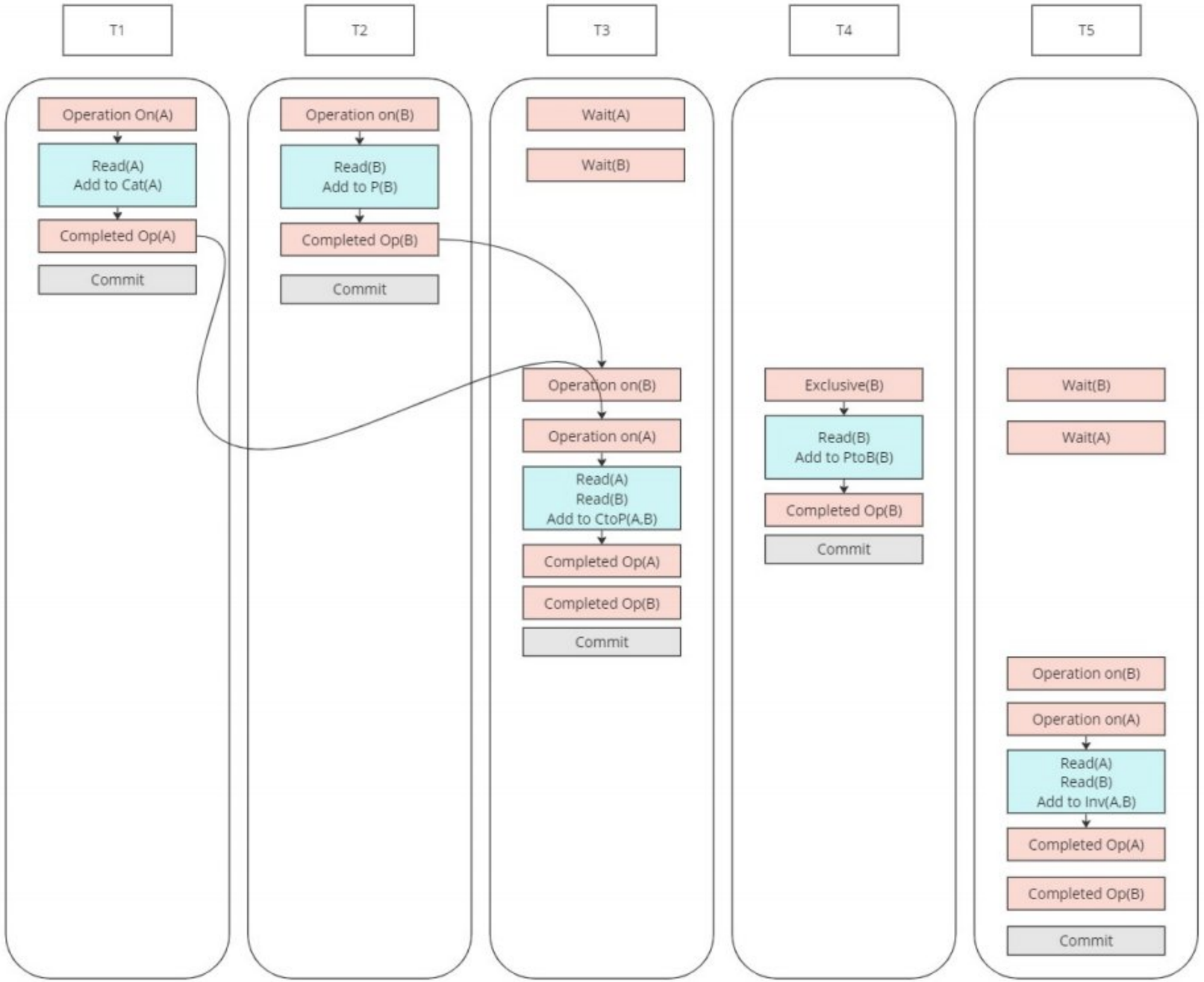
Here the initial quantity of A & B in inventory is **more than** the sum of quantities being ordered by each customer so the order placement times don't matter

Conflict Serializable

Problem Description NCT1

Transaction involves adding a Product B from the store, whose brand exists but its category A is a new one.
Thus first we need to take of adding category first.

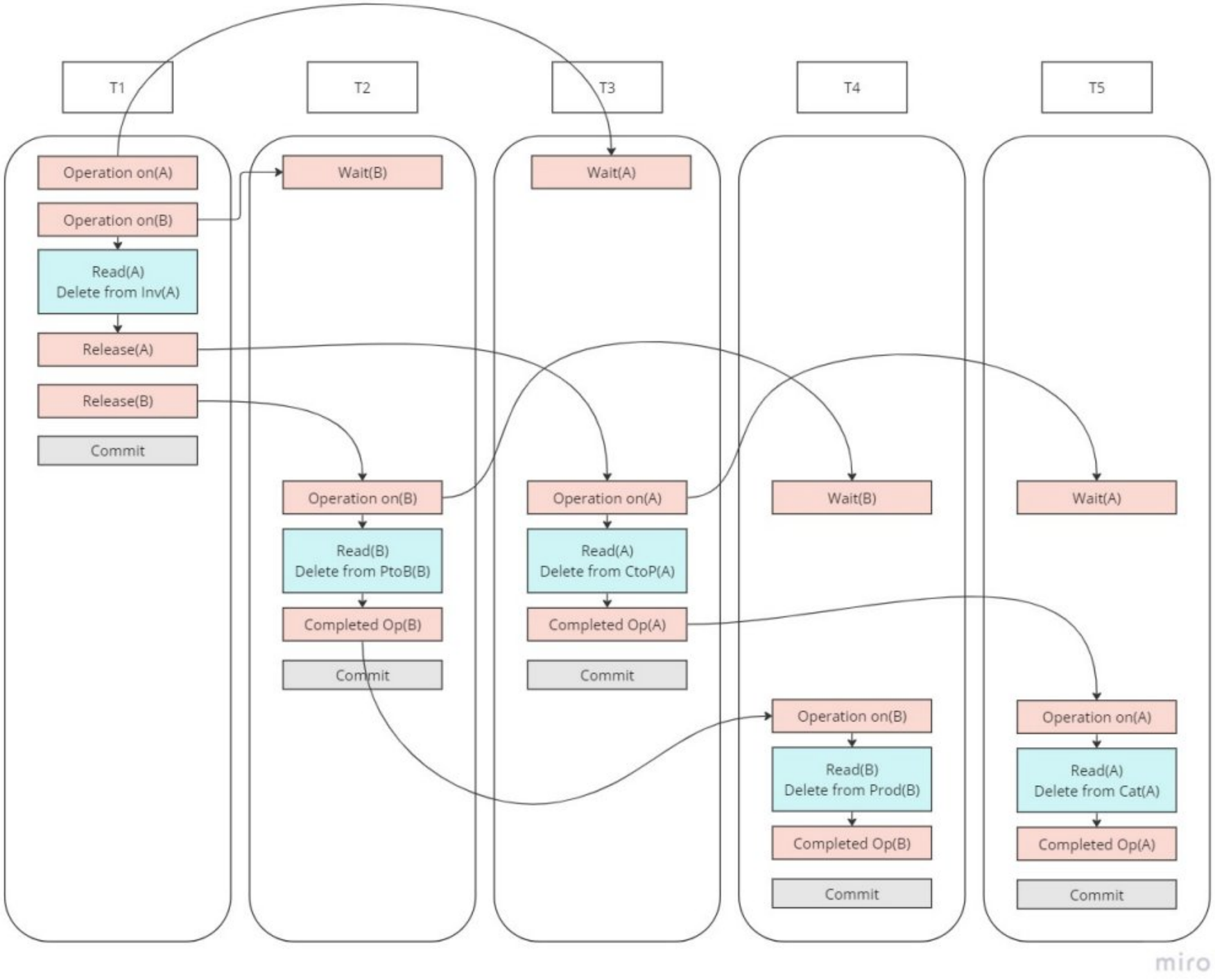
For Non Concurrent operations reason is that we need information of the previous table



Problem Description NCT2

Transaction involves removing a category A from the store, thus entries that feature this category will have to be removed. Also products(Say 1 called B) of this category A will also have to be removed from their respective tables

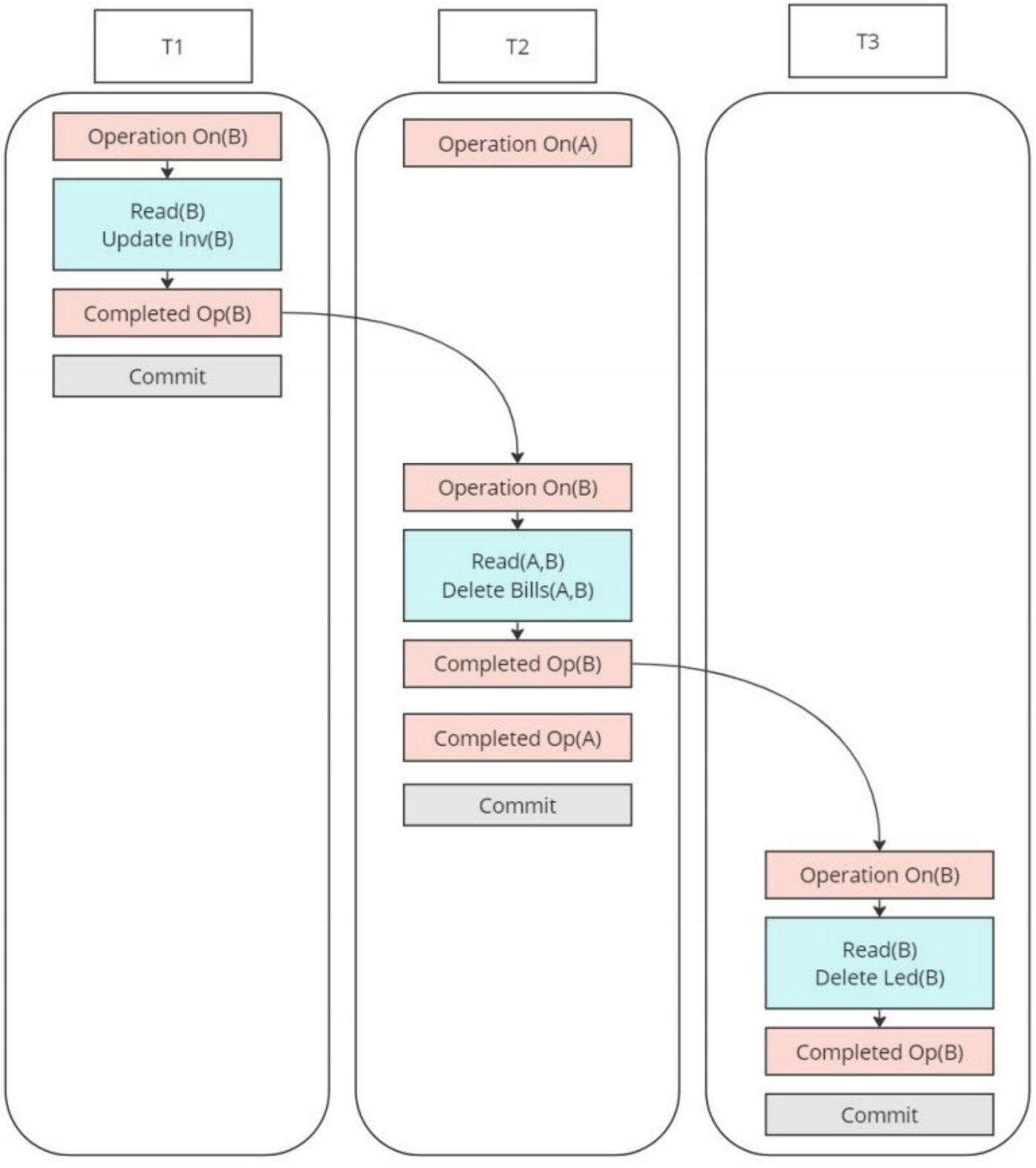
For Non Concurrent operations reason is that we need information of the previous table



Problem Description NCT3

Cancelling an order completely. It not only involves deleting data(A) item from ledger but also from the bills table. In addition we also have to update the quantity of every product(B) of bill(A) in the inventory

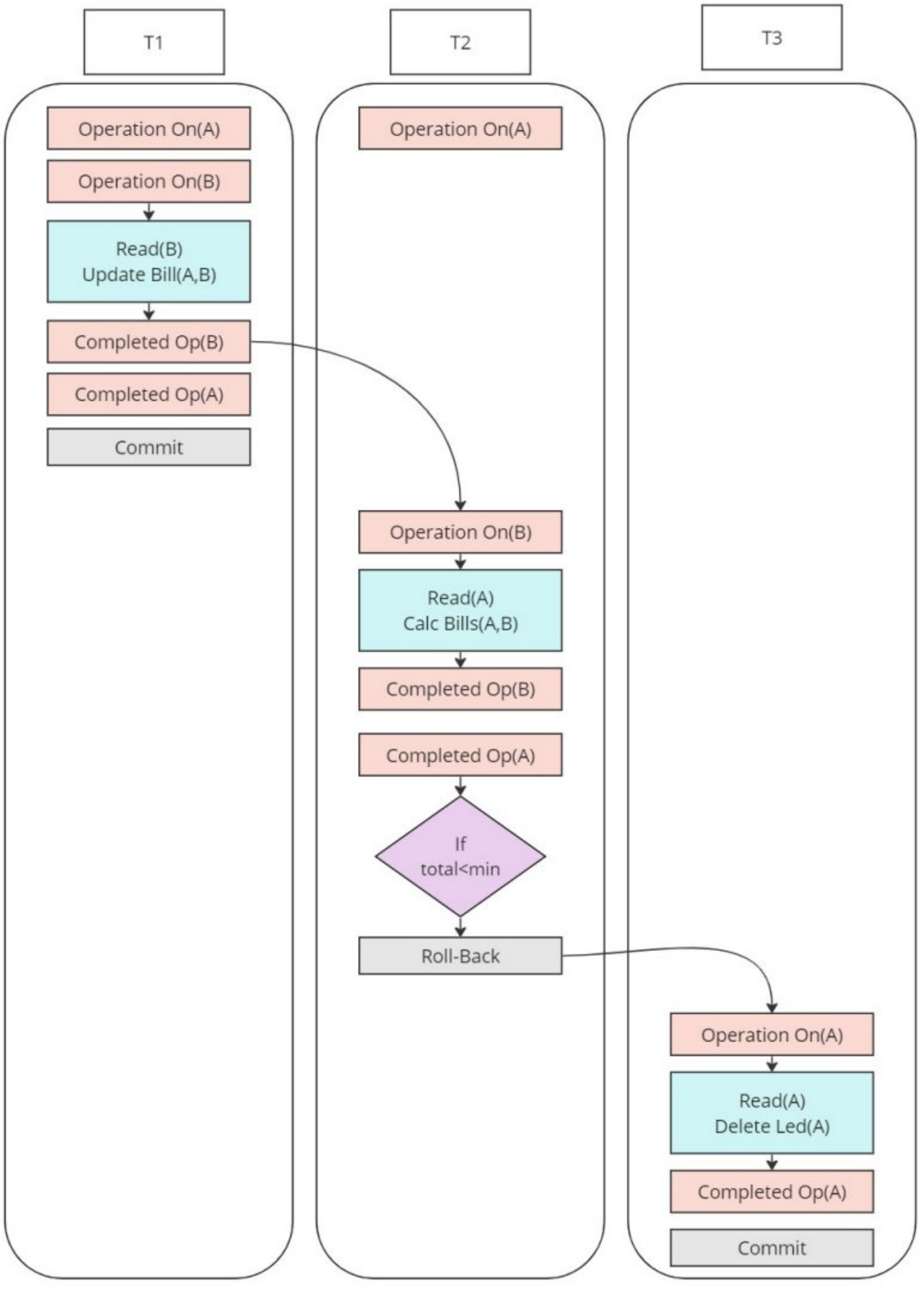
For Non Concurrent operations reason is that we need information of the previous table



Problem Description NCT4

Implementing a minimum order amount requirement.
As soon as bill(A) is generated then it's total is calculated adding each item(B) and if it exceeds the limit then it is committed else it is rolled-back. Here the rollback situation is described

For Non Concurrent operations reason is that we need information of the previous table



SOME OTHER FUNCTIONALITIES OFFERED TO A USER

- Log In / Sign Up with credentials Validity checker
- Add to Cart / View Cart / Empty Cart
- Product View
- Advanced Product Search Function
- Checking one's Order History
- Proceed to Checkout
- Return an Order
- In addition to this, the Admin has the functionality to run any custom query to get the required information for the underlying table.