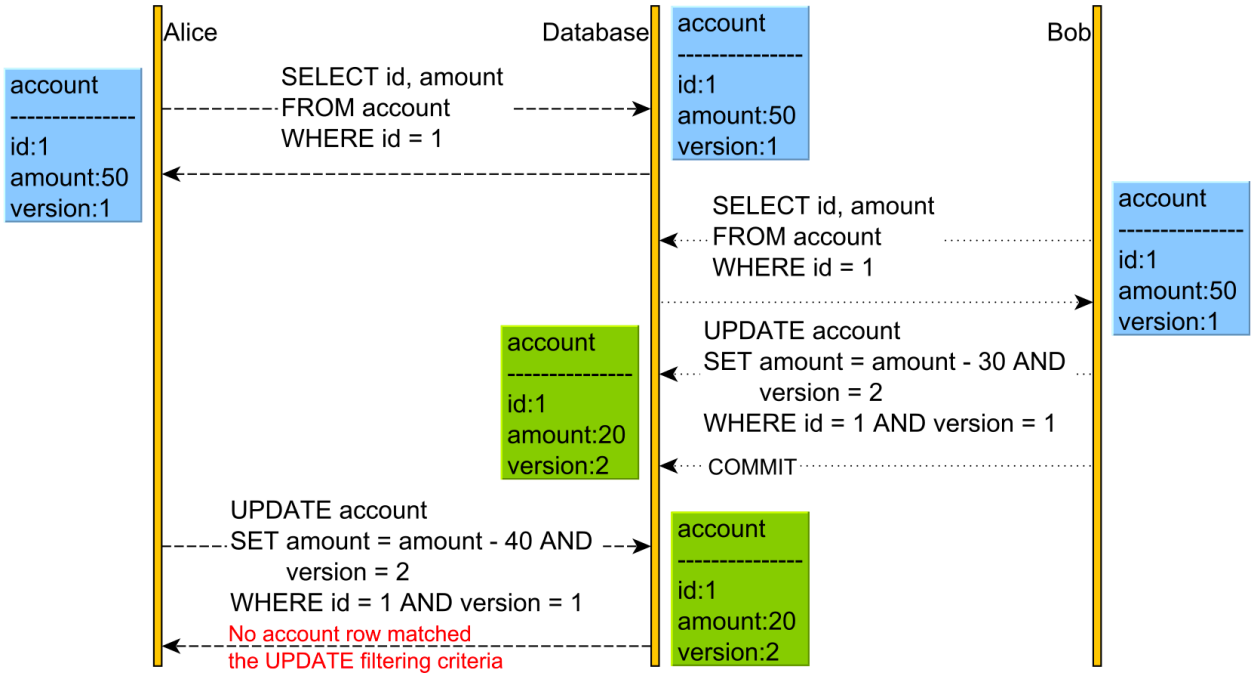
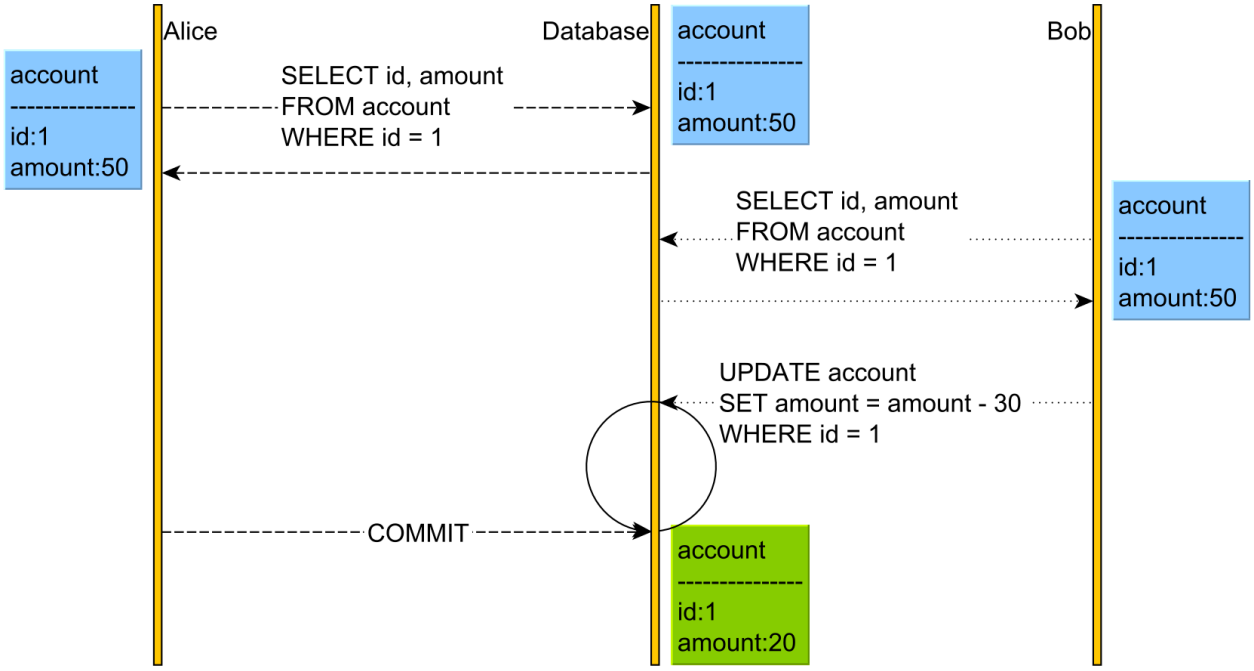
Optimistic lock and pessimistic lock
There are usually two ways to deal with conflicts or collisions, first one is detect and retry it, the second one is avoiding them by blocking concurrent transmitters, like wifi.
Pessimistic locking aims to avoid conflicts by using locking. By using shared lock, none transactions acquired the value can further change it. Any update attempt will be blocked until the lock is released by the holder.
Optimistic locking on the contrary, it allows a conflict to occur, but it needs to detect it at write time , this can be done using either a physical or a logical clock. However, since logical clock is superior to physical clock when it comes to implementing concurrent costal mechanism.
Version number is used to detect it while the other object is reading it.

Example of optimistic locking and pessimistic locking:





Saga
Saga is a design pattern to manage data consistency across micro services in distributed transaction scenario. A saga is a sequence of transactions that updated each service and published a message or event to trigger the next transaction step.
If the step fails, the sage will execute compensating transactions that counteract the preceding transactions.
Another challenge is interprocess communication, synchronicity and availability.

The Saga pattern provides transaction management using a sequence of *local transactions*. A local transaction is the atomic work effort performed by a saga participant. Each local transaction updates the database and publishes a message or event to trigger the next local transaction in the saga. If a local transaction fails, the saga executes a series of *compensating transactions* that undo the changes that were made by the preceding local transactions.

- *Compensable transactions* are transactions that can potentially be reversed by processing another transaction with the opposite effect.
- A *pivot transaction* is the go/no-go point in a saga. If the pivot transaction commits, the saga runs until completion. A pivot transaction can be a transaction that is neither compensable nor retryable, or it can be the last compensable transaction or the first retryable transaction in the saga.
- *Retryable transactions* are transactions that follow the pivot transaction and are guaranteed to succeed.

How to solve deadlock
The only way to resolve a sql server deadlock is to terminate one of the processes and free up the locked resource. So the process can complete. This occur automatically when sql server detects a deadlock and kills of one of the competing process known as victim.
The way to prevent deadlocks are:
1 crate better indexes
2 adjust transaction priorities
3 change isolation modes
4 old locks as short a Tims as possible
…

Where and having

Where clause is used to filter the records from the table or used while joining more than one table. Only those records will be extracted who are satisfying the specified condition in where clause.

Having clause is used to filter the records form groups based on the given condition in the having clause , the gourd satisfying the given condition will appear in the final solutio.
Having clause can only be used with select statement

Where can not contains aggregate function but having could
Where implements in row operation having implements in column
Where can be used with groupBy but having can not
