Homework 8

8.1. optimistic lock and pessimistic lock

What is Optimistic lock?
Optimistic Lock is a strategy where you read a record, and take note of a version number or timestamps or checksums/hashes and check that the version hasn't changed before you write the record back.
If the info(version, sum/hash) is same, just finish writing. If the info is different, should discard the record and can not update(write).

What is pessimistic lock?
Pessimistic lock is when a user starts to update a record, there will be a lock placed on the record. Any others want to update the record simultaneously will be informed that there is an update in process. The other ones should wait until the previous update to be finished(committed).

What is the difference between Optimistic lock and Pessimistic lock?

Optimistic assumes that nothing's going to change while you're reading it. Pessimistic assumes that something will and so locks it.

Unlike pessimistic lock that a user tries to process a record, when the record is locked first, and only the user can operate it until commit, optimistic lock only takes note of a version number when user read a record that does not lock the record. That means the record is allowed to be written(changed) even during the process that there is another one is reading the data.

The scenario of using Optimistic lock:
We assume conflicts happens not too much. We allow the conflict to occur, and the cost of redo a transaction may not be high. Cuz before writing operation, it will has lots of other operations between first reading note and writing operation.

The scenario of using Pessimistic lock:
We try to avoid the conflict at the first time (other transaction try to read a locked record will be declined, and transaction will be ended).

8.2

How to solve the deadlock?
1. Follow a same order to access record, which avoids the loop, but decrease the

performance of concurrence.

2. Avoid the cross accessing, and interaction between two transactions(reduce the source competition)
3. Keep a transaction as short(soon) as possible, which could reduce the occupied time/source

8.3

What is saga design pattern?
Saga is a sequence of local transactions. It works that instead of one transaction in 2PC model, each local transaction updates the data and sends message or even to trigger the next local transaction(other service). If a local transaction fail, the saga will execute a series of compensating transactions to undo the changes(what the preceding local transactions did, look like rollback)

There are two ways:
Choreography: each local transaction to execute the sending message as trigger to next service
Orchestration: an orchestrator to order all the participants(services) what local transactions to execute.