Assignment 3

1. Optimistic lock and pessimistic lock

Optimistic locking is when you check if the record was updated by someone else before you commit the transaction. This strategy is most applicable to high-volume systems and three-tier architectures where you do not necessarily maintain a connection to the database for your session. In this situation the client cannot actually maintain database locks as the connections are taken from a pool and you may not be using the same connection from one access to the next.

Pessimistic locking is when you take an exclusive lock so that no one else can modify the record, which means you lock the record for your exclusive use until you have finished with it. It has much better integrity than optimistic locking but requires you to be careful with your application design to avoid deadlocks.

2. How to prevent deadlock?

- **Conservative 2PL**
  Conservative 2PL transitions obtain all the locks they need before the transactions begin. This is to ensure that a transaction that already holds some locks will not block waiting or other locks.
- **Wait-Die**
  When transaction T1 requests a data item which is currently held by transaction T2, T1 is allowed to wait until the data is available only if T1 is an older transaction than T2, otherwise T1 is killed (die).
- **Wound-Wait**
  This is a counterpart to the Wait-Die Scheme. When transaction T1 requests a data item which is currently held by transaction T2, T1 is allowed to wait until the data is available only if T1 is a younger transaction. Otherwise T2is killed (wounded).

3. Saga design pattern

- **What is Saga design pattern?**

  The Saga pattern is a widely used pattern for distributed transactions. It is different from 2pc, which is synchronous. The Saga pattern is asynchronous and reactive. In a Saga pattern, the distributed transaction is fulfilled by asynchronous local transactions on all related microservices. The microservices communicate with each other through an event bus.If any microservice fails to complete its local transaction, the other microservices will run compensation transactions to rollback the changes.

- **Saga design pattern VS. 2PC**

Saga:

- Pros:
  Supports long-lived transactions. No lock.
- Cons:
  May be difficult to debug. Event messages could be difficult to maintain. Doesn't have read isolation.

2PC:

- Pros:
  2PC is a very strong consistency protocol. Allows read-write isolation. It can guarantee the transaction is atomic.
- Cons:
  2PC is synchronous (blocking), the lock could become a system performance bottleneck.

4. Reference

- https://enterprisecraftsmanship.com/posts/optimistic-locking-automatic-retry/
- https://stackoverflow.com/questions/129329/optimistic-vs-pessimistic-locking
- https://en.wikipedia.org/wiki/Conservative_two-phase_locking
- https://stackoverflow.com/questions/32794142/what-is-the-difference-between-wait-die-and-wound-wait-deadlock-prevention-a
- https://www.geeksforgeeks.org/deadlock-in-dbms/
- https://developers.redhat.com/blog/2018/10/01/patterns-for-distributed-transactions-within-a-microservices-architecture#possible_solutions