Lock, Transaction, SQL

**Optimistic Lock vs Pessimistic Lock**

Optimistic locking is when you check if the record was updated by someone else before you commit the transaction. Pessimistic locking is **when you take an exclusive lock** so that no one else can start modifying the record.

When to use different locks? - depends of the number of the expected collisions. Few -> o, many -> p

**Transaction**

A transaction is **a logical unit of work that contains one or more SQL statements**. A transaction is an atomic unit. The effects of all the SQL statements in a transaction can be either all committed or all rolled back.

**How do we solve a deadlock?**

**Deadlock Characteristics**   
As discussed in the previous post, deadlock has following characteristics. 

1. Mutual Exclusion
2. Hold and Wait
3. No preemption
4. Circular wait

We can prevent Deadlock by eliminating any of the above four conditions.

Bankers’s Algorithm is resource allocation and deadlock avoidance algorithm which test all the request made by processes for resources and to see if it is eligible.

**What is a livelock?**

Livelock occurs when two or more processes continually repeat the same interaction in response to changes in the other processes without doing any useful work. These processes are not in the waiting state, and they are running concurrently. This is different from a deadlock because in a deadlock all processes are in the waiting state.

Distributed Transaction

(A distributed transaction is **a set of operations on data that is performed across two or more data repositories** (especially databases).)

**Saga**

The saga pattern is a failure management pattern that **helps establish consistency in distributed applications.**

A saga is a sequence of transactions that updates each service and publishes a message or event to trigger the next transaction step. If a step fails, the saga executes compensating transactions that counteract the preceding transactions.

**2PC**

the two-phase commit protocol is a type of atomic commitment protocol. It is extremely useful in distributed applications. This protocol is broken into 2 parts: 1) prepare phrase and commit phrase.

**Diagram

Description automatically generated with medium confidence**

**Pro: synchronized, therefore, fault-tolerant**

**Con: The two-phase commit is a blocking protocol; the failure of a single node blocks progress until the node recovers. 2** **if the transaction coordinator fails, then the database is left in an inconsistent state and only recovers once the coordinator recovers. This adds up latency.**