Locking means to maintain SQL server transactions processing successfully and working seamlessly in a multi-user environment. (transaction concurrency) Locking keeps the data integrity for the isolation requirement. (Atomicity, consistency, isolation, durability) Isolation means a transaction is not visible to other transactions until it is complete. A lock can prevent data changes from a transaction.

Optimistic lock

Optimistic locking is a technique for SQL database applications that does not hold row locks between selecting and updating or deleting a row. For optimistic lock, application will assume that unlocked rows are unlikely to change. So, when a row changes, the update or delete will fail, then try select again.

Pros: improve the performance of concurrency.

Cons: result in false positives, requires more re-try logic, complicate to build UPDATE, data type mismatch

Pessimistic lock

Pessimistic lock unlike optimistic lock, always assume the worst case. The worst case is there are always two or more users trying to update the same record simultaneously.

Pros: avoids the conflict (updates are serialized)

How to solve the deadlock

Deadlock: two transaction both need to wait for each other to commit(circle)

1. Obtain information from the lock event monitor about all tables where agents have deadlock.
2. Solve the deadlock
3. Rerun the application

Solutions:

1. Each application connection should process its own set of rows to avoid lock waits.
2. Ensuring that all applications access their common data in the same order
3. A lock timeout
4. Avoid concurrent DDL operations
5. Write actions such as delete, insert, and update
6. Data definition language (DDL) statements, such as ALTER, CREATE, and DROP
7. BIND and REBIND commands

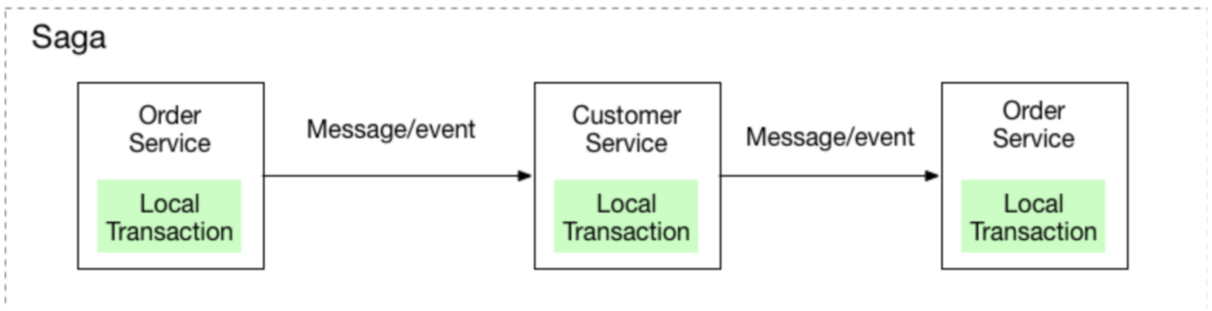
Distributed transaction Saga

Saga: implement each transaction that spans multiple services, a sequence of local transactions. For example, a local transaction tries to update can trigger the next local transaction. If the transaction fails then saga executes compensation transactions to undo the changes.

Two types of Saga:

Choreography - each local transaction publishes domain events that trigger local transactions in other services

Orchestration - an orchestrator (object) tells the participants what local transactions to execute



Having vs where
WHERE cannot used with aggregates, but HAVING can.