Optimistic lock vs. pessimistic lock

Optimistic locking is a technique for SQL database applications that does not hold row locks between selecting and updating or deleting a row. The application is written to optimistically assume that unlocked rows are unlikely to change before the update or delete operation. If the row does change, the update or delete will fail and the application logic handles such failures by retrying the select. It improves concurrency as the other applications can read and write that row.

Multiple users can try to update the same record at the same time, and the record changes are validated only when the record is committed. If one user successfully updates the record, the other users attempting to commit their concurrent updates are informed that a conflict exists.

Pessimistic lock prevents simultaneous updates to record. As soon as user starts to update a record, a lock is placed on it. The other users cannot update this record and have to wait until the first user finished committing.

Optimistic locking: a record is locked only when changes are committed to the database, so multiple users can attempt to update the same record at the same time without noticing.

Pessimistic locking: a record is locked while it is edited, so the other users cannot attempt to update the same record.

For both models, the lock is released after the changes are committed to the database.

Deadlock

Both processes are constantly waiting for each other.

We choose one process to be the deadlock victim and terminate it and then rerun the transaction.

Saga

SQL persistence supports sagas using the core NServiceBus. Saga API or an experimental API unique to SQL persistence that provides a simpler mapping API.

A Saga consists of a sequence of operations, each could work with a resource. Changes made by the operation on the particular resource are visible to the outer world immediately. We can see it as a just group of operations (a.k.a local transactions) which are executed one by one group by the Saga.

A Saga guarantees that either all operations succeed or all the work is undone by compensating actions. The compensating actions are not generically provided by a coordinator framework, instead, they have undone actions defined in business logic by the application programmer.

Two-phase commit

The protocol works in two phases. The first phase is named ‘prepare’ and the coordinator queries participants if they are ready to finish with the commit. The second phase is named ‘commit’ and coordinator commands participants to commit and made changes visible to the outer world. Coordinator commands to commit only if all participants voted for it. If some of the participant votes ‘abort’ then the whole transaction and all participants are rolled back. It means any change made to the participant during the transaction is aborted.