Pattern: Transactional outbox

Also known as

Application events

Context

A service command typically needs to update the database **and** send messages/events. For example, a service that participates in a saga (/patterns/data/saga.html) needs to atomically update the database and sends messages/events. Similarly, a service that publishes a domain event (domain-event.html) must atomically update an aggregate (aggregate.html) and publish an event. The database update and sending of the message must be atomic in order to avoid data inconsistencies and bugs. However, it is not viable to use a distributed transaction that spans the database and the message broker to atomically update the database and publish messages/events.

Problem

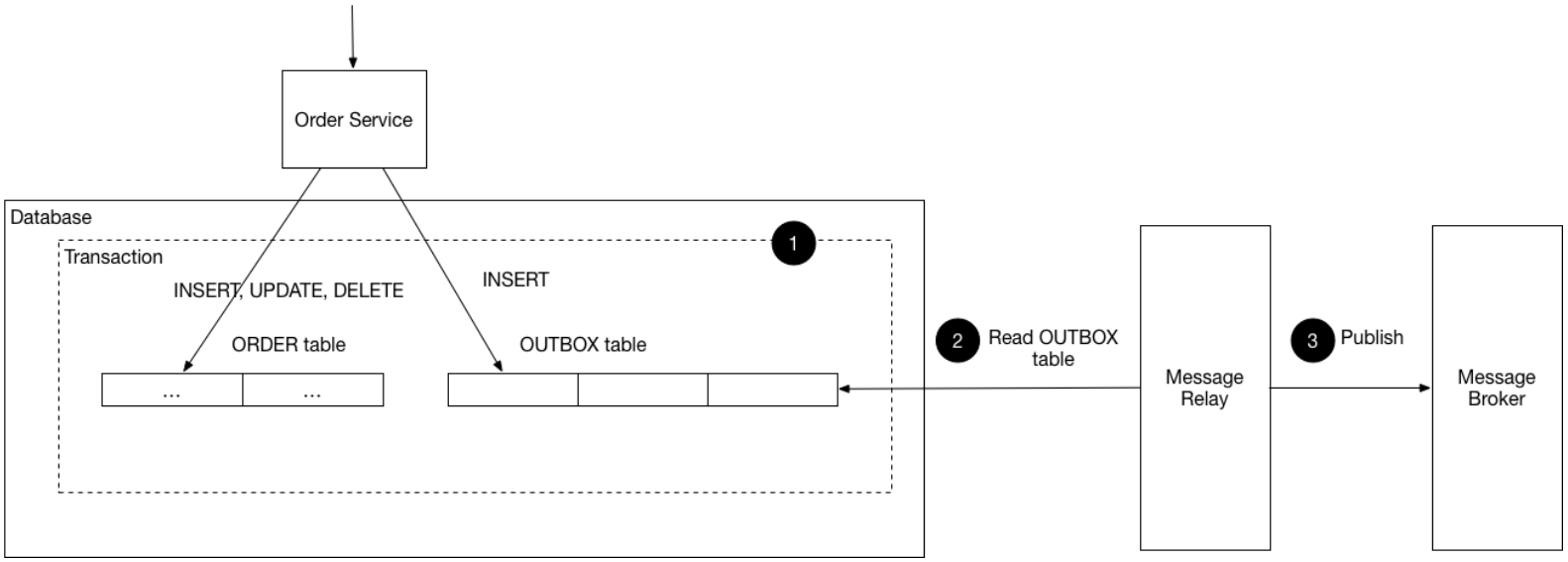
How to reliably/atomically update the database and publish messages/events?

Forces

- 2PC is not an option

Solution

A service that uses a relational database inserts messages/events into an *outbox* table (e.g. `MESSAGE`) as part of the local transaction. An service that uses a NoSQL database appends the messages/events to attribute of the record (e.g. document or item) being updated. A separate *Message Relay* process publishes the events inserted into database to a message broker.



Result context

This pattern has the following benefits:

- The service publishes high-level domain events (domain-event.html)
- No 2PC required

This pattern has the following drawbacks:

- Potentially error prone since the developer might forget to publish the message/event after updating the database.

This pattern also has the following issues:

- The Message Relay might publish a message more than once. It might, for example, crash after publishing a message but before recording the fact that it has done so. When it restarts, it will then publish the message again. As a result, a message consumer must be idempotent, perhaps by tracking the IDs of the messages that it has already processed. Fortunately, since Message Consumers usually need to be idempotent (because a message broker can deliver messages more than once) this is typically not a problem.

Related patterns

- The Saga (saga.html) and Domain event (domain-event.html) patterns create the need for this pattern.
- The Event sourcing (event-sourcing.html) is an alternative solution
- There are two patterns for implementing the message relay:
    - The Transaction log tailing (transaction-log-tailing.html) pattern
    - The Polling publisher (polling-publisher.html) pattern

Learn more

- My book Microservices patterns (/book) describes this pattern in a lot more detail.
- The Eventuate Tram framework (https://github.com/eventuate-tram/eventuate-tram-core) implements this pattern