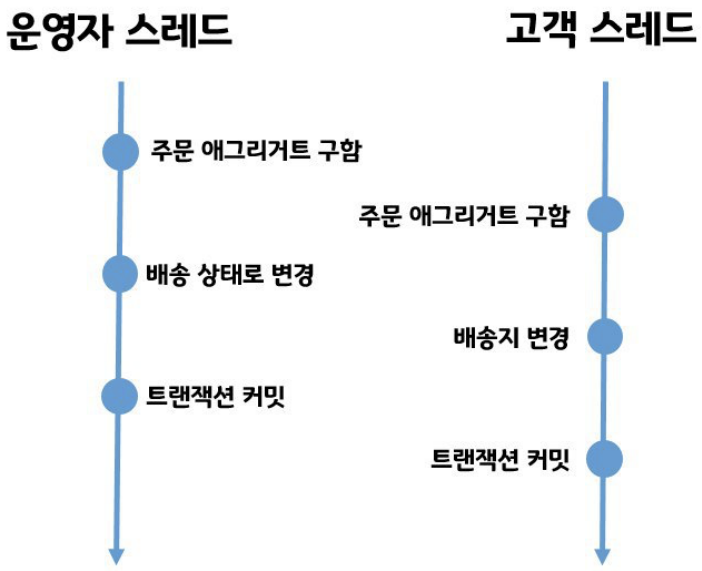
08. 애그리거트 트랜잭션 관리

version 2018.35

애그리거트와 트랜잭션

한 주문 애그리거트에 대해 운영자가 배송 상태로 변경한다고 가정합시다. 그런데 변경 도중 고객이 주문 애그리거트에 대해 배송지 주소를 변경하면 어떻게 될까요? 다음 그림은 운영자와 고객이 동시에 한 주문 애그리거트를 수정하는 과정을 보여줍니다.

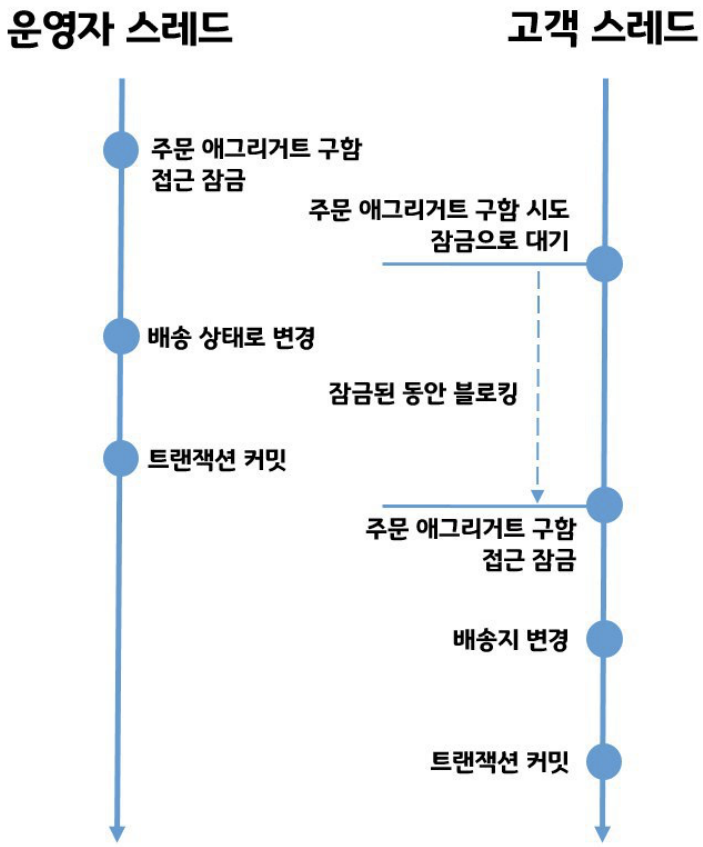


> 메모리 캐시를 사용하지 않을 경우, 두 스레드는 개념적으로 동일한 애그리거트이지만 물리적으로는 서로 다른 애그리거트 객체를 사용합니다.

이 경우 **'배송 상태(배송 중 or 배송 완료)에는 배송지를 변경할 수 없다'**라는 비즈니스 로직을 만족할 수 없게 되므로 애그리거트의 일관성이 깨집니다. 이런 문제가 발생하지 않도록 하려면 DBMS에서 지원하는 트랜잭션과 함께 애그리거트를 위한 추가적인 트랜잭션 처리 기법이 필요합니다.

선점 잠금

애그리거트에 대해 사용할 수 있는 대표적인 트랜잭션 처리 방식에는 **선점(Pessimistic)** 잠금과 **비선점(Optimistic)** 잠금이 있습니다. 먼저 선점을 알아보겠습니다. **선점 잠금(Pressimistic Lock)**은 먼저 애그리거트를 구한 스레드의 애그리거트 사용이 끝날 때까지 다른 스레드가 해당 애그리거트를 수정하는 것을 막는 방식입니다. 앞서 배송 상태 변경 문제에 선점 잠금을 적용하면 아래의 그림과 같습니다.



선점 잠금은 보통 DBMS가 제공하는 행 단위 잠금을 사용해서 구현합니다. 오라클을 비롯한 다수 DBMS가 for update와 같은 쿼리를 사용해서 특정 레코드에 한 사용자만 접근할 수 있는 잠근 장치를 제공합니다.

> for update

선점 잠금과 교착 상태

선점 잠금 기능을 사용할 때는 잠금 순서에 따른 **교착 상태(deadlock)**가 발생하지 않도록 주의해야 합니다. 이런 문제가 발생하지 않도록 하기 위해 잠금을 구할 때 최대 대기 시간을 지정해야 합니다.

DBMS에 따라 교착 상태에 빠진 커넥션을 처리하는 방식이 다릅니다. 쿼리별로 대기 시간을 지정할 수 있는 DBMS가 있고 커넥션 단위로만 대기 시간을 지정할 수 있는 DBMS도 있습니다. 따라서, 선점 잠금을 사용하기전에 DBMS에 대해 조사 한 후에 사용해야합니다.

비선점 잠금

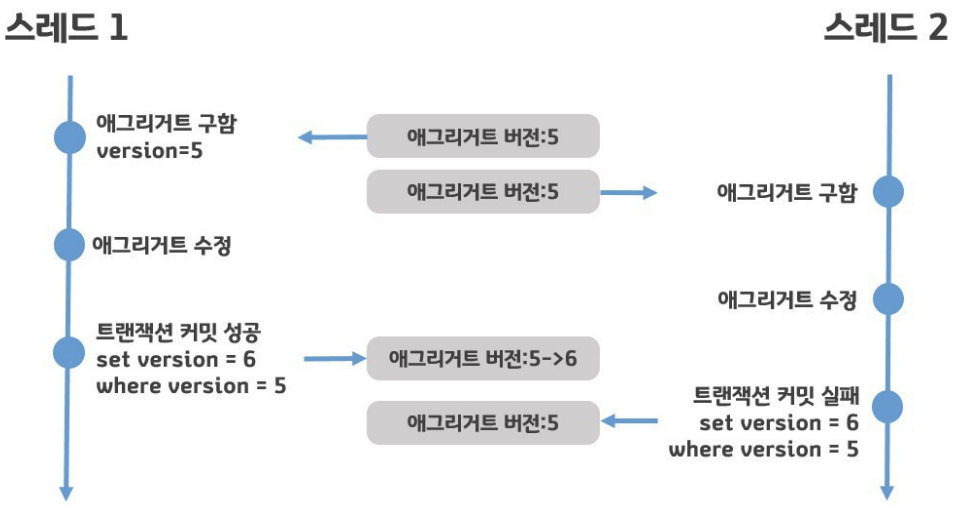
선점 잠금이 모든 트랜잭션 충돌 문제를 해결 할 수 있는 것은 아닙니다. 아래의 그림은 선점 잠금으로 해결 할 수 없는 상황을 보여줍니다.

이 처럼 선점 잠금 방식으로 해결 할 수 없는 문제를 **비선점 잠금(Optimistic Lock)**으로 해결 할 수 있습니다. 비선점 잠금 방식은 변경한 데이터를 실제 DBMS에 반영하는 시점에 변경 가능 여부를 확인하는 방식입니다.

비선점 잠금을 구현하려면 애그리거트에 버전으로 사용할 숫자 타입의 프로퍼티를 추가 해야합니다. 애그리거트를 수정할 때마다 버전으로 사용할 프로퍼티의 값이 1씩 증가 하는데, 이때 다음과 같은 쿼리를 사용합니다.

```
UPDATE [aggtable] SET version = version + 1, [colum_x] = ?
WHERE [aggid] = ? and version = 현재버전
```

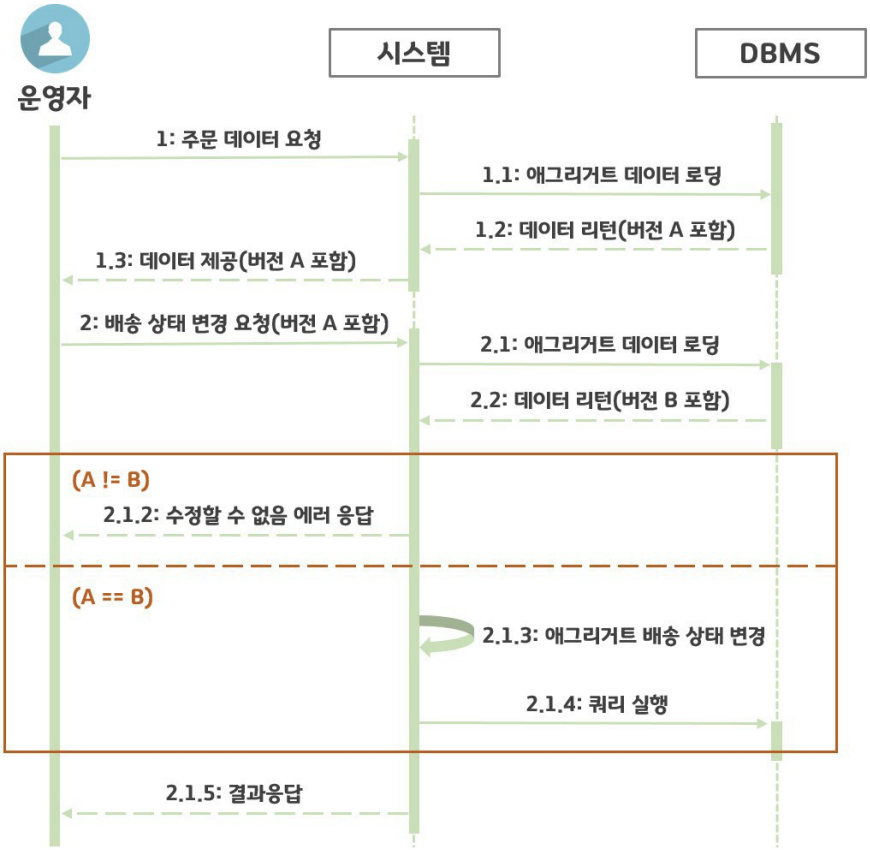
이 쿼리는 수정할 애그리거트의 버전 값이 현재 애그리거트의 버전과 동일한 경우에만 데이터를 수정합니다. 이를 그림으로 표현하면 아래과 같습니다.



비선점 잠금을 위한 쿼리를 실행할 때 쿼리 실행 결과로 수정된 행의 개수가 0이면 이미 누군가 앞서 데이터를 수정한 것입니다. 이는 트랜잭션이 충돌한 것이므로 트랜잭션 종료 시점에 익셉션이 발생 시킵니다. 스프링에서는 OptimisticLockingFailure-Exception이 있습니다.

비선점 잠금의 확장

비선점 잠금을 아래의 그림 처럼 **확장하여** 트랙잭션 충돌 문제를 해소할 수도 있습니다.



구현

그림 처럼 비선점 잠금 방식을 여러 트랜잭션으로 확장하려면 애그리거트 정보를 뷰로 보여줄 때 버전 정보도 함께 사용자 화면에 전달해야합니다. 아래의 코드 처럼 버전 값을 갖는 hidden 타입 `<input>` 태그 사용해서 서버에 폼 전송 시 버전 값이 함께 전달 되도록합니다.

```
<form action="startShipping" method="POST">
    <input type="hidden" name="version" value="{$orderDto.version}">
    <input type="text" name="orderNumber" value="{$orderDto.orderNumber}"
readonly>
    ...
    <input type="submit" value="배송 상태로 변경하기">
</form>
```

배송 상태 변경을 처리하는 응용 서비스가 전달받는 데이터는 다음과 같이 주문 번호화 함께 해당 주문을 조회한 시점의 버전 값을 포함해야 합니다.

```
public class StartShippingRequest {

    private String orderNumber;
    private long version;

    ...생성자, getter
}
```

응용 서비스는 전달받은 버전 값을 이용해서 애그리거트의 버전과 일치하는지 확인 하고 일치하는 경우에만 요청한 기능을 수행합니다.

```
public class StartShippingService {

    @PreAuthorize("hasRole('ADMIN')")
    @Transactional
    public void startShipping(StartShippingRequest req) {
        Order order = orderRepository.findById(new
OrderNo(req.getOrderNumber()));
        checkOrder(order);
        if (!order.matchVersion(req.getVersion())) {
            throw new VersionConflictException();
        }
        order.startShipping();
    }
    ...
}
```

```
@Controller
public class OrderAdminController {

    private StartShippingService startShippingService;

    @RequestMapping(value = "/startShipping", method = RequestMethod.POST)
    public String startShipping(StartShippingRequest startReq) {
        try {
            startShippingService.startShipping(startReq);
            return "shippingStarted";
        } catch(VersionConflictException |
OptimisticLockingFailureException ex) {
            // 트랜잭션 충돌
            return "startShippingTxConflict";
        }
    }
    ...
}
```

이 코드는 비선점 잠금과 관련해서 발생하는 두 개의 익셉션을 처리하고 있습니다. **두 익셉션은 발생시키는 위치가 다릅니다.** VersionConflictExeption은 응용 서비스 코드에서 발생키키고, OptimisticLockingFailureException는 스프링 프레임워크가 발생시킵니다.

> VersionConflictExeption은 이미 누군가가 애그리거트를 수정했다는 것을 의미
>
> OptimisticLockingFailureException은 누군가가 거의 동시에 애그리거트를 수정했다는 것을 의미

강제 버전 증가

애그리거트에 애그리거트 루트 외에 다른 엔티티가 존재하는데 기능 실행 도중 루트가 아닌 다른 엔티티의 값만 변경된다고 해봅시다. 비록 루트 엔티티의 값이 바뀌지 않았더라도 애그리거트의 구성요소 중 일부 값이 바뀌면 논리적으로 그 애그리거트는 바뀐 것이다. 따라서, **애그리거트 내에 어떤 구성요소의 상태가 바뀌면 루트 애그리거트의 버전 값을 증가해야한다.**

오프라인 선점 잠금

오프라인 선점 잠금을 위한 LockManager 인터페이스와 관련 클래스

DB를 이용한 LockManager 구현