STEP09 - Concurrency Report

🔍1. 분석: 내 서비스의 동시성 문제

시나리오: e-커머스 서비스

다수의 고객이 동시에 특정 "인기 상품"을 구매할 경우, 해당 상품의 재고와 관련된 데이터가 동시에 변경될 수 있습니다. 이때 동시 업데이트로 인해 데이터 정합성이 깨지는 문제가 발생할 수 있습니다.

동시성 문제 1: 상품 재고 차감

- **문제 발생 조건**: 동시에 여러 사용자가 같은 상품을 구매 시도할 경우
- **현재 코드 시나리오**:

```
Item item = itemRepository.findById(itemId).orElseThrow(...);
item.decrease(stock); // -1
itemRepository.save(item);
```

- **문제점**: 여러 트랜잭션에서 동시에 같은 상품을 조회하면 재고 차감 전 상태의 데이터가 공유되어 재고 초과 사용이 발생할 수 있습니다.

동시성 문제 2: 유저 포인트 차감

- **현재 코드 시나리오**:

```
User user = userRepository.findById(userId).orElseThrow(...);
user.decreasePoint(amount); // -500
userRepository.save(user);
```

- **문제점**: 여러 요청에서 같은 시점에 포인트를 차감하려 하면 사용자의 실제 포인트보다 많은 차감이 발생할 수 있습니다.

동시성 문제 3: 선착순 쿠폰 발급

- **현재 상황**: 쿠폰이 선착순으로 여러 명에게 동시에 발급 가능
- **문제점**: 동시에 여러 사용자가 쿠폰 발급을 시도할 경우, 발급 수량을 초과해 발급되는 문제가 발생할 수 있습니다.

⚖️2. 해결책: DB 레벨의 동시성 제어 기능 활용

해결 1: 상품 재고 - Pessimistic Lock (비관적 락)

- `@Lock(LockModeType.PESSIMISTIC_WRITE)`를 JPA Repository 메서드에 적용
- **예시 코드**:

```
@Lock(PESSIMISTIC_WRITE)
@Query("select i from Item i where i.id = :id")
Optional<Item> findByIdWithLock(@Param("id") Long id);
```

해결 2: 유저 포인트 - JPQL 조건 업데이트

- `update User set point = point - :amount where id = :userId and point >= :amount`
- 조건식으로 포인트가 부족할 경우 차감이 수행되지 않도록 제어함

해결 3: 쿠폰 발급 - Optimistic Lock (낙관적 락)

- `@Version` 필드를 포함하여 버전 체크를 통해 충돌 감지
- 충돌 시 `OptimisticLockException` 발생 → 재시도 전략 필요

🔗테스트 및 결과

상품 재고

- 100명의 사용자가 동시에 다른 상품을 구매하는 시나리오를 테스트
- 정확한 재고 차감과 저장이 이루어졌으며, 재고 초과 사용 없음 확인

유저 포인트

- JPQL 조건식을 활용하여 포인트가 부족한 경우 차감되지 않도록 함
- 테스트 결과 유효성 100% 유지 및 동시성 오류 없음

쿠폰 발급

- 10명의 사용자가 동시에 쿠폰 발급 요청 시도
- 1\~2건의 `OptimisticLockException` 발생 후 재시도 로직을 통해 정해진 수량만 발급 완료됨
- 복수 발급 방지 성공

후기

- 서비스의 특성에 따라 적절한 DB Lock 전략을 선택하는 것이 중요함
- 전체적으로 "동시 요청으로 인한 문제"들을 분석하고, 어떤 방식으로 제어할지 고민하게 되었음
- 공통 기준이 되는 동시성 제어 전략을 정립하고, 트랜잭션 경계와 책임 분리를 명확히 하는 설계가 중요함