

콘서트 예약 서비스에 있는 기능 중 동시성 문제가 발생할 수 있는 로직은 좌석예약과 잔액 관련 기능입니다.

- 좌석예약

```
@Transactional
public Optional<Seat> reservation(long scheduleId, long seatNumber) throws Exception {
    log.info("[스레드ID : {}] 서비스 시작!!", Thread.currentThread().getId());

    Optional<Seat> seatForReservation = seatRepository.getSeatForReservation(scheduleId, seatNumber); //lock 획득
    if (!seatForReservation.isPresent()) {
        log.error("[스레드ID : {}] 해당 좌석 없음!!", Thread.currentThread().getId());
        throw new ResourceNotFoundException("해당 좌석 없음");
    }

    log.info("[스레드ID : {}] 예약 락 획득!!", Thread.currentThread().getId());
    seatForReservation.get().reservation(); //예약처리 (좌석임시, update불파 갱신)

    Optional<Seat> saveSeat = seatRepository.save(seatForReservation.get());
    log.info("[스레드ID : {}] 예약처리 완료!!", Thread.currentThread().getId());

    if (!saveSeat.isPresent()) {
        throw new ResourceNotFoundException("좌석 예약 실패");
    }
    log.info("[스레드ID : {}] 서비스 끝!!", Thread.currentThread().getId());
    return saveSeat;
}
```

먼저 좌석 예약은 위와 같이 되어 있으며
seatRepository.getSeatForReservation() 을 통해 유효성 검사를 진행 후,
좌석을 임시예약처리 상태로 save() 하는 로직 입니다.

좌석 예약 동시성 테스트는 아래 로직으로 진행하였습니다.

```
@Test
void 좌석예약_동시성테스트() throws Exception {
    Thread.sleep( millis 1000); //BeforeEach 도는 시간 기다림
    int numThreads = 100; //스레드 개수
    long seatNumber = 6L;

    CountDownLatch latch = new CountDownLatch(numThreads); //스레드들을 동시 시작 및 종료를 관리하기 위한 객체
    ExecutorService executorService = Executors.newFixedThreadPool(numThreads);

    Long startTime = System.currentTimeMillis();

    for (int i = 0; i < numThreads; i++) {
        executorService.submit() -> {
            try {
                concertFacade.reservation(scheduleId, seatNumber);
            } catch (ObjectOptimisticLockingFailureException e) {
                log.error("[스레드ID : {}] ObjectOptimisticLockingFailureException :: {}", Thread.currentThread().getId(), e.getMessage());
            } catch (Exception ex) {
                log.error("[스레드ID : {}] Exception :: {}", Thread.currentThread().getId(), ex.getMessage());
            } finally {
                latch.countDown();
            }
        }
    }

    latch.await(); // 모든 스레드가 완료될 때까지 대기
    executorService.shutdown(); //스레드 풀 종료

    Long endTime = System.currentTimeMillis();
    log.info("소요 시간: {}", (endTime - startTime) + "ms");
}
```

1. 낙관락

> version 을 통해 update 를 하는 쿼리

```
Hibernate: update seat set schedule_id=?,seat_number=?,status=?,updated_at=?,version=? where id=? and version=?
```

>> 동시 100건 테스트

총 소요시간 : 93ms

동시성 이슈 : 9건

> 정상 작동 로그 (1건)

```
2024-07-25T20:44:14.733+09:00 INFO 31408 --- [concertreservation] [pool-2-thread-7] c.h.c.application.facade.ConcertFacade : [쓰레드ID : 35] 파사드 시작!!
2024-07-25T20:44:14.734+09:00 INFO 31408 --- [concertreservation] [pool-2-thread-7] c.h.c.domain.concert.ConcertService : [쓰레드ID : 35] 서비스 시작!!
2024-07-25T20:44:14.782+09:00 INFO 31408 --- [concertreservation] [pool-2-thread-7] c.h.c.domain.concert.ConcertService : [쓰레드ID : 35] 예약 락 획득!!
2024-07-25T20:44:14.785+09:00 INFO 31408 --- [concertreservation] [pool-2-thread-7] c.h.c.domain.concert.ConcertService : [쓰레드ID : 35] 예약처리 완료!!
2024-07-25T20:44:14.785+09:00 INFO 31408 --- [concertreservation] [pool-2-thread-7] c.h.c.domain.concert.ConcertService : [쓰레드ID : 35] 서비스 끝!!
2024-07-25T20:44:14.787+09:00 INFO 31408 --- [concertreservation] [pool-2-thread-7] c.h.c.application.facade.ConcertFacade : [쓰레드ID : 35] 파사드 끝!!
```

정상작동하여 "파사드 끝" 로그까지 찍혀있습니다.

> 동시성 에러 로그 (ex 1건)

```
2024-07-25T20:44:14.731+09:00 INFO 31408 --- [concertreservation] [pool-2-thread-2] c.h.c.application.facade.ConcertFacade : [쓰레드ID : 30] 파사드 시작!!
2024-07-25T20:44:14.733+09:00 INFO 31408 --- [concertreservation] [pool-2-thread-2] c.h.c.domain.concert.ConcertService : [쓰레드ID : 30] 서비스 시작!!
2024-07-25T20:44:14.783+09:00 INFO 31408 --- [concertreservation] [pool-2-thread-2] c.h.c.domain.concert.ConcertService : [쓰레드ID : 30] 예약 락 획득!!
2024-07-25T20:44:14.785+09:00 INFO 31408 --- [concertreservation] [pool-2-thread-2] c.h.c.domain.concert.ConcertService : [쓰레드ID : 30] 예약처리 완료!!
2024-07-25T20:44:14.785+09:00 INFO 31408 --- [concertreservation] [pool-2-thread-2] c.h.c.domain.concert.ConcertService : [쓰레드ID : 30] 서비스 끝!!
2024-07-25T20:44:14.805+09:00 ERROR 31408 --- [concertreservation] [pool-2-thread-2] c.h.c.i.ConcertIntegrationTest : [쓰레드ID : 30] ObjectOptimisticLockingFailureException :: Row was updated or deleted by another transaction (or unsaved-value mapping was incorrect) : [con.hh.concertreservation.infra.concert.SeatEntity#6]
```

파사드에서 시작하여 서비스까지 로직을 타고 들어갔으며,

save() 때 바로 ObjectOptimisticLockingFailureException 이 터지지 않는 것을 보아 Version 정보가 맞지 않아 동시성 에러가 터질 경우 save() 할때 바로 터지는게 아니라 서비스가 다 돌고(트랜잭션이 끝나고) 터지는 것을 확인할 수 있었습니다.

>> 동시 1500건 테스트

총 소요시간 : 716ms

동시성 이슈 : 10건

> 정상 작동 로그 (1건)

```
2024-07-25T21:11:16.300+09:00 INFO 70356 --- [concertreservation] [pool-2-thread-5] c.h.c.application.facade.ConcertFacade : [쓰레드ID : 33] 파사드 시작!!
2024-07-25T21:11:16.301+09:00 INFO 70356 --- [concertreservation] [pool-2-thread-5] c.h.c.domain.concert.ConcertService : [쓰레드ID : 33] 서비스 시작!!
2024-07-25T21:11:16.348+09:00 INFO 70356 --- [concertreservation] [pool-2-thread-5] c.h.c.domain.concert.ConcertService : [쓰레드ID : 33] 예약 락 획득!!
2024-07-25T21:11:16.357+09:00 INFO 70356 --- [concertreservation] [pool-2-thread-5] c.h.c.domain.concert.ConcertService : [쓰레드ID : 33] 예약처리 완료!!
2024-07-25T21:11:16.357+09:00 INFO 70356 --- [concertreservation] [pool-2-thread-5] c.h.c.domain.concert.ConcertService : [쓰레드ID : 33] 서비스 끝!!
2024-07-25T21:11:16.410+09:00 INFO 70356 --- [concertreservation] [pool-2-thread-5] c.h.c.application.facade.ConcertFacade : [쓰레드ID : 33] 파사드 끝!!
```

정상작동하여 "파사드 끝" 로그까지 찍혀있습니다.

> 동시성 에러 로그 (ex 1건)

```
2024-07-25T21:11:16.301+09:00 INFO 70356 --- [concertreservation] [pool-2-thread-12] c.h.c.application.facade.ConcertFacade : [쓰레드ID : 40] 파사드 시작!!
2024-07-25T21:11:16.301+09:00 INFO 70356 --- [concertreservation] [pool-2-thread-12] c.h.c.domain.concert.ConcertService : [쓰레드ID : 40] 서비스 시작!!
2024-07-25T21:11:16.352+09:00 INFO 70356 --- [concertreservation] [pool-2-thread-12] c.h.c.domain.concert.ConcertService : [쓰레드ID : 40] 예약 락 획득!!
2024-07-25T21:11:16.356+09:00 INFO 70356 --- [concertreservation] [pool-2-thread-12] c.h.c.domain.concert.ConcertService : [쓰레드ID : 40] 예약처리 완료!!
2024-07-25T21:11:16.356+09:00 INFO 70356 --- [concertreservation] [pool-2-thread-12] c.h.c.domain.concert.ConcertService : [쓰레드ID : 40] 서비스 끝!!
2024-07-25T21:11:16.442+09:00 ERROR 70356 --- [concertreservation] [pool-2-thread-12] c.h.c.i.ConcertIntegrationTest : [쓰레드ID : 40] ObjectOptimisticLockingFailureException :: Row was updated or deleted by another transaction (or unsaved-value mapping was incorrect) : [con.hh.concertreservation.infra.concert.SeatEntity#6]
```

100건 테스트와 마찬가지로 트랜잭션이 끝난 뒤 exception 이 터진걸 확인할 수 있었습니다.

2. 비관락

> 락 획득하기위해 날리는 쿼리

```
Hibernate: select se1_0.id,se1_0.schedule_id,se1_0.seat_number,se1_0.status,se1_0.updated_at,se1_0.version from seat se1_0 where se1_0.schedule_id=? and se1_0.seat_number=? and se1_0.status='EMPTY' for update
```

>> 동시 100건 테스트

총 소요시간 : 93ms

동시성 이슈 : 0건

> 정상 작동 로그 (1건)

```
2024-07-25T21:32:29.902+09:00 INFO 34020 --- [concertreservation] [ool-2-thread-13] c.h.c.application.facade.ConcertFacade : [쓰레드ID : 42] 파사드 시작!!
2024-07-25T21:32:29.902+09:00 INFO 34020 --- [concertreservation] [ool-2-thread-13] c.h.c.domain.concert.ConcertService : [쓰레드ID : 42] 서비스 시작!!
2024-07-25T21:32:29.934+09:00 INFO 34020 --- [concertreservation] [ool-2-thread-13] c.h.c.domain.concert.ConcertService : [쓰레드ID : 42] 예약 락 획득!!
2024-07-25T21:32:29.937+09:00 INFO 34020 --- [concertreservation] [ool-2-thread-13] c.h.c.domain.concert.ConcertService : [쓰레드ID : 42] 예약처리 완료!!
2024-07-25T21:32:29.937+09:00 INFO 34020 --- [concertreservation] [ool-2-thread-13] c.h.c.domain.concert.ConcertService : [쓰레드ID : 42] 서비스 끝!!
2024-07-25T21:32:29.956+09:00 INFO 34020 --- [concertreservation] [ool-2-thread-13] c.h.c.application.facade.ConcertFacade : [쓰레드ID : 42] 파사드 끝!!
```

> 정상 작동 로그 (실패처리 99건)

```
Exception :: 해당 좌석 없음
2024-07-25T21:32:29.967+09:00 ERROR 34020 --- [concertreservation] [ool-2-thread-22] c.h.c.i.ConcertIntegrationTest : [쓰레드ID : 51] Exception :: 해당 좌석 없음
2024-07-25T21:32:29.967+09:00 ERROR 34020 --- [concertreservation] [ool-2-thread-23] c.h.c.i.ConcertIntegrationTest : [쓰레드ID : 52] Exception :: 해당 좌석 없음
2024-07-25T21:32:29.967+09:00 ERROR 34020 --- [concertreservation] [ool-2-thread-15] c.h.c.i.ConcertIntegrationTest : [쓰레드ID : 44] Exception :: 해당 좌석 없음
2024-07-25T21:32:29.967+09:00 ERROR 34020 --- [concertreservation] [ool-2-thread-29] c.h.c.i.ConcertIntegrationTest : [쓰레드ID : 58] Exception :: 해당 좌석 없음
2024-07-25T21:32:29.968+09:00 ERROR 34020 --- [concertreservation] [ool-2-thread-26] c.h.c.i.ConcertIntegrationTest : [쓰레드ID : 55] Exception :: 해당 좌석 없음
2024-07-25T21:32:29.968+09:00 ERROR 34020 --- [concertreservation] [pool-2-thread-6] c.h.c.i.ConcertIntegrationTest : [쓰레드ID : 34] Exception :: 해당 좌석 없음
2024-07-25T21:32:29.969+09:00 ERROR 34020 --- [concertreservation] [ool-2-thread-24] c.h.c.i.ConcertIntegrationTest : [쓰레드ID : 53] Exception :: 해당 좌석 없음
2024-07-25T21:32:29.969+09:00 ERROR 34020 --- [concertreservation] [ool-2-thread-32] c.h.c.i.ConcertIntegrationTest : [쓰레드ID : 61] Exception :: 해당 좌석 없음
```

>> 동시 1500건 테스트

총 소요시간 : 764ms

동시성 이슈 : 0건

> 정상 작동 로그 (1건)

```
2024-07-25T22:36:32.700+09:00 INFO 11272 --- [concertreservation] [pool-2-thread-8] c.h.c.application.facade.ConcertFacade : [쓰레드ID : 36] 파사드 시작!!
2024-07-25T22:36:32.700+09:00 INFO 11272 --- [concertreservation] [pool-2-thread-8] c.h.c.domain.concert.ConcertService : [쓰레드ID : 36] 서비스 시작!!
2024-07-25T22:36:32.748+09:00 INFO 11272 --- [concertreservation] [pool-2-thread-8] c.h.c.domain.concert.ConcertService : [쓰레드ID : 36] 예약 락 획득!!
2024-07-25T22:36:32.753+09:00 INFO 11272 --- [concertreservation] [pool-2-thread-8] c.h.c.domain.concert.ConcertService : [쓰레드ID : 36] 예약처리 완료!!
2024-07-25T22:36:32.753+09:00 INFO 11272 --- [concertreservation] [pool-2-thread-8] c.h.c.domain.concert.ConcertService : [쓰레드ID : 36] 서비스 끝!!
2024-07-25T22:36:32.782+09:00 INFO 11272 --- [concertreservation] [pool-2-thread-8] c.h.c.application.facade.ConcertFacade : [쓰레드ID : 36] 파사드 끝!!
```

> 정상 작동 로그 (실패처리 1499건)

```
Exception :: 해당 좌석 없음
2024-07-25T22:36:33.433+09:00 ERROR 11272 --- [concertreservation] [l-2-thread-1491] c.h.c.domain.concert.ConcertService : [쓰레드ID : 1520] 해당 좌석 없음!!
2024-07-25T22:36:33.433+09:00 ERROR 11272 --- [concertreservation] [l-2-thread-1494] c.h.c.domain.concert.ConcertService : [쓰레드ID : 1523] 해당 좌석 없음!!
2024-07-25T22:36:33.433+09:00 ERROR 11272 --- [concertreservation] [l-2-thread-1493] c.h.c.domain.concert.ConcertService : [쓰레드ID : 1522] 해당 좌석 없음!!
2024-07-25T22:36:33.433+09:00 ERROR 11272 --- [concertreservation] [l-2-thread-1492] c.h.c.domain.concert.ConcertService : [쓰레드ID : 1521] 해당 좌석 없음!!
2024-07-25T22:36:33.433+09:00 ERROR 11272 --- [concertreservation] [l-2-thread-1496] c.h.c.domain.concert.ConcertService : [쓰레드ID : 1525] 해당 좌석 없음!!
```

- 정리

쓰레드 개수	낙관적 락	비관적 락
100 개	93ms	93ms
1500 개	716ms	764ms

100건으로 진행했을 때는 큰 차이가 없었지만, 그 수가 많아질수록 속도 차이가 나는 것 처럼 보입니다. 좌석예약의 기능은 1건만 성공이 된다면 나머지 요청들은 모두 실패처리 해도 되는 기능이고, 낙관적 락은 그 쓰레드 수가 많을수록 비관적 락에 비해 성능적인 부분에서 더 차이를 보일 것입니다. 따라서 해당 기능은 낙관적 락이 적합하다고 생각합니다.

- 잔액충전

```
@Transactional
public Optional<UserBalance> charge(long userId, long amount) throws Exception {
    log.info("[스레드ID : {}] 서비스 시작!", Thread.currentThread().getId());

    Optional<UserBalance> result = userBalanceRepository.findByUserIdWithLock(userId);
    log.info("[스레드ID : {}] 충전 락 획득!", Thread.currentThread().getId());

    if (result.isPresent()) {
        UserBalance userBalance = result.get();
        userBalance.charge(amount);

        Optional<UserBalance> balance = userBalanceRepository.save(userBalance);
        if (balance.isPresent()) {
            log.info("[스레드ID : {}] 충전 완료!", Thread.currentThread().getId());
            log.info("[스레드ID : {}] 서비스 끝!", Thread.currentThread().getId());
            return balance;
        }
    }

    log.info("[스레드ID : {}] 서비스 끝!", Thread.currentThread().getId());
    return Optional.empty();
}
```

충전은 위와같이 되어있으며 findByUserIdWithLock() 으로 락 획득 후 charge() 로 포인트 충전후 save() 처리하는 로직 입니다.

잔액충전 동시성테스트는 아래 로직으로 진행하였습니다.

```
@Test
void 충전_동시성테스트() throws Exception {
    long amount = 1000L;
    int numThreads = 100;    //스레드 개수
    Optional<UserBalance> beforeBalance = cashFacade.getUserBalance(userId);

    CountDownLatch latch = new CountDownLatch(numThreads); //스레드들을 동시 시작 및 종료를 관리하기 위한 객체
    ExecutorService executorService = Executors.newFixedThreadPool(numThreads); //정해진 스레드들(numThreads)에게 동시에 작업할당을 하기위한 객체

    Long startTime = System.currentTimeMillis();

    for (int i = 0; i < numThreads; i++) {
        executorService.submit(() -> {
            try {
                cashFacade.charge(userId, amount);
            } catch (Exception ex) {
                log.error("[스레드ID : {}] Exception :: {}", Thread.currentThread().getId(), ex.getMessage());
            } finally {
                latch.countDown();
            }
        });
    }

    latch.await(); // 모든 스레드가 완료될 때까지 대기
    executorService.shutdown(); //스레드 풀 종료

    Long endTime = System.currentTimeMillis();
    log.info("소요 시간: {}", (endTime - startTime) + "ms");

    Optional<UserBalance> balance = cashFacade.getUserBalance(userId);
    log.info("before Balance : {}", beforeBalance.get().getBalance());
    log.info("충전한 포인트 : {} 원씩 {} 번 = {}", amount, numThreads, amount * numThreads);
    log.info("after Balance : {}", balance.get().getBalance());
    Assertions.assertEquals(balance.get().getBalance(), actual: beforeBalance.get().getBalance() + amount * numThreads);
}
```


1. 낙관락

> version 을 통해 update 를 하는 쿼리

```
Hibernate: update user_balance set balance=?,updated_at=?,version=? where user_id=? and version=?
```

>> 동시 100건 테스트

총 소요시간 : 251ms

동시성 이슈 : 99건, ObjectOptimisticLockingFailureException 처리된 건은 83건.

좌석예약 기능과 달리 충전 기능은 단순히 데이터가 계속 update 되는 형식이라, (잔액 데이터에 대해 조회를 통한 유효성검사 로직이 불가능) 동시성 테스트를 대량으로 하면 일정 동시성 요청에 대해서는 ObjectOptimisticLockingFailureException 처리를 하지만, 테스트 결과 100건 중 16건은 그대로 충전이 되는 것을 알 수 있었습니다.

사용자 입장에서 충전 기능을 100건 이상 할 케이스가 없을 뿐더러, 좌석예약 기능 처럼 데이터 검증에 대한 유효성검사가 안되는 경우 충전 기능에 대해서는 3~5건에 대한 동시성 테스트만 진행하면 될 것 같다고 생각했습니다.

>> 동시 5건 테스트

총 소요시간 : 68ms

동시성 이슈 : 4건, ObjectOptimisticLockingFailureException 처리된 건 4건

> 충전 결과 (시도는 100원씩 5건이 동시에 시도, 결과는 100원만 충전)

```
2024-07-26T01:29:12.870+09:00 INFO 68628 --- [consertreservation] [ main] c.h.c.integration.CashIntegrationTest : before Balance : 2020615
2024-07-26T01:29:12.870+09:00 INFO 68628 --- [consertreservation] [ main] c.h.c.integration.CashIntegrationTest : 충전한 포인트 : 100 원씩 5 번 = 500
2024-07-26T01:29:12.870+09:00 INFO 68628 --- [consertreservation] [ main] c.h.c.integration.CashIntegrationTest : after Balance : 2020715
```

> ObjectOptimisticLockingFailureException 4건

```
2024-07-26T01:29:12.866+09:00 ERROR 68628 --- [consertreservation] [pool-2-thread-5] c.h.c.integration.CashIntegrationTest : [쓰레드ID : 33] ObjectOptimisticLockingFailureException :: Row was updated or deleted by another transaction (or unsaved-value mapping was incorrect) : [com.hh.consertreservation.infra.cash.UserBalanceEntity#1]
2024-07-26T01:29:12.866+09:00 ERROR 68628 --- [consertreservation] [pool-2-thread-2] c.h.c.integration.CashIntegrationTest : [쓰레드ID : 30] ObjectOptimisticLockingFailureException :: Row was updated or deleted by another transaction (or unsaved-value mapping was incorrect) : [com.hh.consertreservation.infra.cash.UserBalanceEntity#1]
2024-07-26T01:29:12.866+09:00 ERROR 68628 --- [consertreservation] [pool-2-thread-4] c.h.c.integration.CashIntegrationTest : [쓰레드ID : 32] ObjectOptimisticLockingFailureException :: Row was updated or deleted by another transaction (or unsaved-value mapping was incorrect) : [com.hh.consertreservation.infra.cash.UserBalanceEntity#1]
2024-07-26T01:29:12.866+09:00 ERROR 68628 --- [consertreservation] [pool-2-thread-1] c.h.c.integration.CashIntegrationTest : [쓰레드ID : 29] ObjectOptimisticLockingFailureException :: Row was updated or deleted by another transaction (or unsaved-value mapping was incorrect) : [com.hh.consertreservation.infra.cash.UserBalanceEntity#1]
```

2. 비관락

> 락 획득하기 위해 날리는 쿼리

```
Hibernate: select ube1_0.user_id,ube1_0.balance,ube1_0.updated_at from user_balance ube1_0 where ube1_0.user_id=? for update
```

>> 동시 100건 테스트

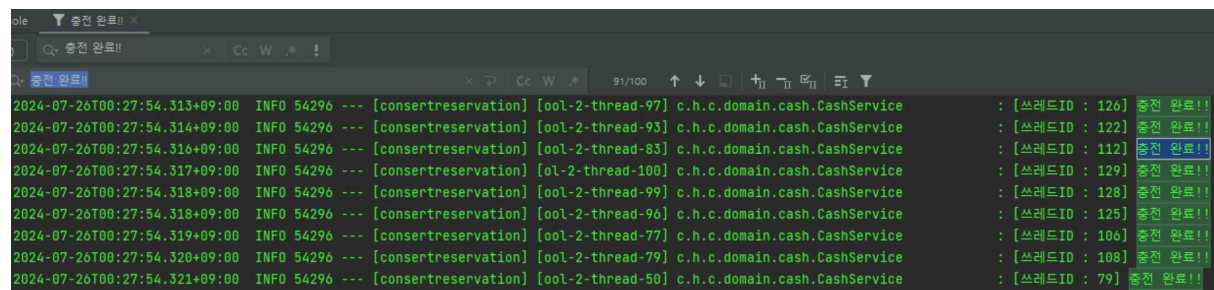
총 소요시간 : 334ms

동시성 이슈 : 0건

> 결과

```
2024-07-26T00:27:54.333+09:00 INFO 54296 --- [consertreservation] [    main] c.h.c.integration.CashIntegrationTest : before Balance : 241015
2024-07-26T00:27:54.333+09:00 INFO 54296 --- [consertreservation] [    main] c.h.c.integration.CashIntegrationTest : 충전한 포인트 : 1000 원씩 100 번 = 100000
2024-07-26T00:27:54.333+09:00 INFO 54296 --- [consertreservation] [    main] c.h.c.integration.CashIntegrationTest : after Balance : 341015
```

> 충전완료 로그 100개



```
2024-07-26T00:27:54.313+09:00 INFO 54296 --- [consertreservation] [ool-2-thread-97] c.h.c.domain.cash.CashService : [쓰레드ID : 126] 충전 완료!!
2024-07-26T00:27:54.314+09:00 INFO 54296 --- [consertreservation] [ool-2-thread-93] c.h.c.domain.cash.CashService : [쓰레드ID : 122] 충전 완료!!
2024-07-26T00:27:54.316+09:00 INFO 54296 --- [consertreservation] [ool-2-thread-83] c.h.c.domain.cash.CashService : [쓰레드ID : 112] 충전 완료!!
2024-07-26T00:27:54.317+09:00 INFO 54296 --- [consertreservation] [ol-2-thread-100] c.h.c.domain.cash.CashService : [쓰레드ID : 129] 충전 완료!!
2024-07-26T00:27:54.318+09:00 INFO 54296 --- [consertreservation] [ool-2-thread-99] c.h.c.domain.cash.CashService : [쓰레드ID : 128] 충전 완료!!
2024-07-26T00:27:54.318+09:00 INFO 54296 --- [consertreservation] [ool-2-thread-96] c.h.c.domain.cash.CashService : [쓰레드ID : 125] 충전 완료!!
2024-07-26T00:27:54.319+09:00 INFO 54296 --- [consertreservation] [ool-2-thread-77] c.h.c.domain.cash.CashService : [쓰레드ID : 106] 충전 완료!!
2024-07-26T00:27:54.320+09:00 INFO 54296 --- [consertreservation] [ool-2-thread-79] c.h.c.domain.cash.CashService : [쓰레드ID : 108] 충전 완료!!
2024-07-26T00:27:54.321+09:00 INFO 54296 --- [consertreservation] [ool-2-thread-50] c.h.c.domain.cash.CashService : [쓰레드ID : 79] 충전 완료!!
```

>> 동시 1500건 테스트

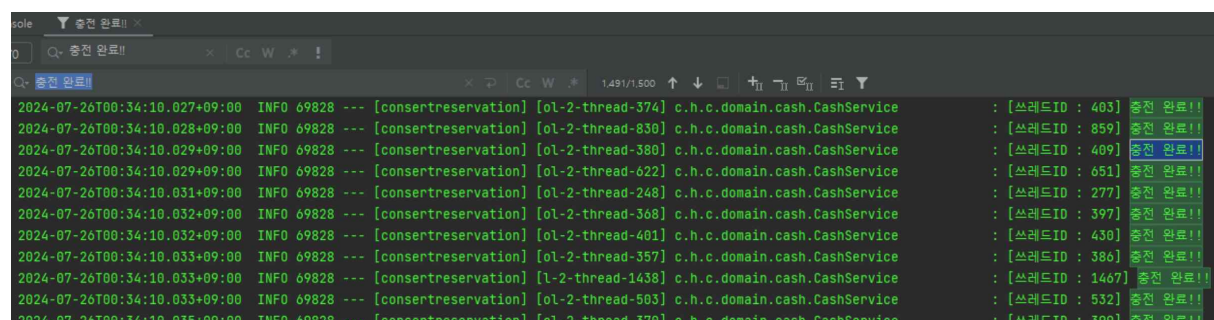
총 소요시간 : 3606ms

동시성 이슈 : 0건

> 결과

```
2024-07-26T00:34:10.086+09:00 INFO 69828 --- [consertreservation] [    main] c.h.c.integration.CashIntegrationTest : before Balance : 1841015
2024-07-26T00:34:10.086+09:00 INFO 69828 --- [consertreservation] [    main] c.h.c.integration.CashIntegrationTest : 충전한 포인트 : 100 원씩 1500 번 = 150000
2024-07-26T00:34:10.087+09:00 INFO 69828 --- [consertreservation] [    main] c.h.c.integration.CashIntegrationTest : after Balance : 1991015
```

> 충전완료 로그 1500개



```
2024-07-26T00:34:10.027+09:00 INFO 69828 --- [consertreservation] [ol-2-thread-374] c.h.c.domain.cash.CashService : [쓰레드ID : 403] 충전 완료!!
2024-07-26T00:34:10.028+09:00 INFO 69828 --- [consertreservation] [ol-2-thread-830] c.h.c.domain.cash.CashService : [쓰레드ID : 859] 충전 완료!!
2024-07-26T00:34:10.029+09:00 INFO 69828 --- [consertreservation] [ol-2-thread-380] c.h.c.domain.cash.CashService : [쓰레드ID : 409] 충전 완료!!
2024-07-26T00:34:10.029+09:00 INFO 69828 --- [consertreservation] [ol-2-thread-622] c.h.c.domain.cash.CashService : [쓰레드ID : 651] 충전 완료!!
2024-07-26T00:34:10.031+09:00 INFO 69828 --- [consertreservation] [ol-2-thread-248] c.h.c.domain.cash.CashService : [쓰레드ID : 277] 충전 완료!!
2024-07-26T00:34:10.032+09:00 INFO 69828 --- [consertreservation] [ol-2-thread-368] c.h.c.domain.cash.CashService : [쓰레드ID : 397] 충전 완료!!
2024-07-26T00:34:10.032+09:00 INFO 69828 --- [consertreservation] [ol-2-thread-401] c.h.c.domain.cash.CashService : [쓰레드ID : 430] 충전 완료!!
2024-07-26T00:34:10.033+09:00 INFO 69828 --- [consertreservation] [ol-2-thread-357] c.h.c.domain.cash.CashService : [쓰레드ID : 386] 충전 완료!!
2024-07-26T00:34:10.033+09:00 INFO 69828 --- [consertreservation] [l-2-thread-1438] c.h.c.domain.cash.CashService : [쓰레드ID : 1467] 충전 완료!!
2024-07-26T00:34:10.033+09:00 INFO 69828 --- [consertreservation] [ol-2-thread-503] c.h.c.domain.cash.CashService : [쓰레드ID : 532] 충전 완료!!
2024-07-26T00:34:10.035+09:00 INFO 69828 --- [consertreservation] [ol-2-thread-370] c.h.c.domain.cash.CashService : [쓰레드ID : 360] 충전 완료!!
```

- 정리

쓰레드 개수	낙관적 락	비관적 락
5 개	68ms	-
100 개	251ms	334ms
1500 개	-	3606ms

충전 기능의 경우 요청이 사용자의 클릭 이슈 등으로 2~3번 동시에 들어올 경우에 대비해서 최초 1건 처리후 그 뒤에 동시에 들어온 요청들은 실패로 처리해야 합니다. 비관적 락의 경우 모든 요청을 처리해버리니 이 기능에 맞지 않는다고 생각하여 낙관적 락이 적합하다고 생각하였습니다.

- 결제

> 현재 제 프로젝트 상에서는 이미 해당 콘서트의 결제정보가 있는지 확인 후 결제요청 데이터를 insert 하는 형식 입니다. 따라서 낙관적 락의 목적인 version 정보 를 통해 데이터 감지 목적과는 다르고, 불가능하다고 생각하여 비관적 락이 적합하다고 생각하였습니다.