본 어플리케이션에서는 이벤트성 사업으로 쿠폰발급 이벤트를 진행한다.

- 우측 상단에 쿠폰 이벤트 목록을 클릭하면 쿠폰 이벤트 페이지로 넘어가고,

- 리스트된 이벤트 정보 중 원하는 이벤트 쿠폰의 쿠폰 받기 버튼을 클릭하면 남은 수량에 따라 사용자에게 쿠폰이 발급된다.
- 만약, 해당 이벤트 쿠폰의 잔여 수량이 1개인 상태에서, 여러 명의 사용자가 동시에 해당 쿠폰 받기를 누른다면 여러 개의 트랜잭션이 수행될 것이다.
- 이때, 해당 이벤트 쿠폰의 잔여 수량이 0개로 바뀌더라도 이미 여러 명의 사용자가 잔여 수량이 1개인 상태에서 동시에 트랜잭션에 진입한 상태이기에 잔여 수량이 0개로 바뀌었음에도 이를 인지하지 못하고 동일한 쿠폰이 잔여 수량을 초과하여 발급되게 된다.

Concurrency Control Solution

```
class CouponModel {
  insert = async (couponEventId: number, userId: number) => {
    const conn = await database.getConnection();
    await conn.execute("SET TRANSACTION ISOLATION LEVEL SERIALIZABLE");
    const quantitySql = `SELECT * FROM COUPON_EVENT WHERE COUPON_EVENT_ID = ${couponEventId}`;
    const quantity = (await conn.execute<CouponEventDto>(quantitySql)).rows?.[0]
      .QUANTITY;
    if (quantity && quantity > 0) {
      const autoIncrement = `(SELECT NVL(MAX(COUPON_ID), 0) + 1 FROM COUPON)`;
      const couponSql = `INSERT INTO COUPON VALUES (${autoIncrement}, ${userId}, '미사용', ${couponEventId})`;
      const couponEventSql = `UPDATE COUPON_EVENT SET QUANTITY = ${
        quantity - 1
      } WHERE COUPON_EVENT_ID = ${couponEventId}`;
      try {
        await conn.execute(couponSql);
        await conn.execute(couponEventSql);
        await conn.commit();
      } catch (err) {
        console.error(err);
        if (isDBError(err)) {
          await conn.rollback();
        }
        throw err;
      }
    } else {
      throw new Error("쿠폰이 모두 소진되었습니다.");
    }
  };
```

- 이벤트 쿠폰 발급 트랜잭션을 SERIALIZABLE 하도록 ISOLATION LEVEL을 설정한다.
- 먼저 처리된 선행 트랜잭션에 의해 COUPON table은 LOCK에 걸린다.
- 그 외 후행 트랜잭션들은 모두 액세스가 제한되고 선행 트랜잭션의 commit 이후 ORA-08177로 인한 DB Error로 rollback을 진행한다. 즉 데이터베이스 일관성, 무결성 에러를 방지하는 작업을 수행한다.
- 선행 트랜잭션에 의해 남은 쿠폰이 0개로 업데이트 되면 해당 이벤트 쿠폰은 모든 발급이 완료돼 이벤트 대상에서 제외된다.