트랜잭션이 무엇이고, ACID 원칙은 무엇인지 설명해 주세요.

트랜잭션이란?

> 데이터베이스의 상태를 변경하는 하나의 논리적 작업 단위
>
> 즉, 여러 작업을 **모두 성공하거나 모두 실패**시켜 데이터 일관성을 유지하는 단위

예시

은행 계좌 이체 예시

A 계좌에서 10만 원 출금

B 계좌로 10만 원 입금

둘 다 성공해야 이체가 완료. 하나라도 실패하면 전체 취소(rollback)

ACID 원칙(트랜잭션의 속성)

원칙	의미	예시
A - Atomicity (원자성)	모두 수행되거나 전혀 수행되지 않아야 함	이체 중 실패 시 전체 롤백
C - Consistency (일관성)	트랜잭션 전후로 DB 무결성 유지	전체 잔액 일정
I - Isolation (고립성)	여러 트랜잭션이 동시에 실행되어도 간섭 없음	동시에 출금해도 정확한 잔액 반영
D - Durability (지속성)	커밋된 데이터는 장애 발생 후에도 유지	커밋 후 서버 꺼져도 데이터 유지

ACID 원칙 중, Durability는 DBMS를 어떻게 보장하나요?

보상 트랜잭션이 실패하면?

① 재시도(Retry with Backoff)

보상 트랜잭션이 실패하면, 일정 시간 후 **자동 재시도**

성공할 때까지 반복 (exponential backoff)

네트워크 불안정, 임시 장애 등에 효과적

```
// Pseudocode while (!success) { try { cancelPayment(); success = t
rue; } catch (Exception e) { Thread.sleep(2000); // 재시도 } }
```

Saga Coordinator 또는 **메시지 큐**(Kafka, SQS) 가 재시도 관리

---

② Dead Letter Queue (DLQ) + 관리자 보정(Manual Compensation)

여러 번 재시도했음에도 실패하면,

→ 실패 이벤트를 Dead Letter Queue에 저장

이후 **운영자나 관리자**가 수동으로 보정

예시:

"결제 취소 실패" 이벤트를 DLQ로 전송

운영자는 관리자 페이지에서 해당 거래를 확인 후 직접 취소

실무에선 거의 필수로 이 보정 절차를 둠.

---

③ 보상 트랜잭션의 멱등성(Idempotency) 확보

보상 로직은 **여러 번 호출돼도 결과가 동일해야 함**.

왜냐하면 재시도 중 중복 실행될 수 있기 때문

```
@Transactional public void cancelPayment(Long paymentId) { Paymen
t p = paymentRepository.findById(paymentId); if (p.getStatus().equa
ls("CANCELED")) return; // 이미 취소됨 p.setStatus("CANCELED"); paym
entRepository.save(p); }
```

멱등성 보장이 재시도 전략의 전제 조건

---

④ 보상 트랜잭션의 '보상 트랜잭션'

일부 아키텍처에서는

> 보상 트랜잭션이 실패했을 때의 복구 절차(Secondary Compensation)를 별도로 둔다.

예시:

결제 취소가 실패하면, **고객 포인트를 다시 적립**하는 "역보상 트랜잭션" 수행

단, 이건 매우 복잡하기 때문에 일반적으로는 수동 보정으로 해결

트랜잭션을 사용해 본 경험이 있나요? 어떤 경우에 사용할 수 있나요?

데이터 일관성 보장

금융 거래

쇼핑 카트 결제

데이터 무결성 유지

학생 등록 시스템

재고 관리 시스템

동시성 제어

멀티유저 환경: 여러 사용자가 동시에 DB에 접근하여 데이터를 읽고 쓰는 경우, 트랜잭션을 사용해서 데이터 일관성을 유지한다. 예를 들어, 두 사용자가 동시에 같은 제품을 구매할 때, 트랜잭션을 통해 재고가 정확히 관리되어야 한다.

업데이트 충돌 방지

복구 및 오류 처리

시스템 장애 복구

오류 처리

복잡한 데이터 변경

배치 처리: 대규모 데이터 변경 작업은 여러 단계로 구성되며, 각 단계가 모두 완료되지 않으면 모두 취소된다.

데이터 마이그레이션: DB 스키마 변경이나 데이터 이전 작업을 수행할 때, 트랜잭션을 사용하여 모든 변경 사항이 올바르게 적용되었는지 확인한다.

읽기에는 트랜잭션을 걸지 않아도 될까요?

읽기(SELECT)에는 트랜잭션을 걸지 않아도 될까요?

> 대부분의 경우 읽기 전용 작업에는 트랜잭션이 필요하지 않음
>
> 하지만 **일관된 조회**(Consistent Read) 가 필요한 경우에는 트랜잭션을 걸기도 함

트랜잭션을 걸지 않아도 되는 경우

단순 조회 (예: 게시글 목록, 단일 사용자 정보 등)

캐시나 조회 전용 API에서 읽기만 수행할 때

트랜잭션이 필요한 경우

보고서 생성, 정산 등 **동일 시점의 데이터 일관성이 필요한 경우**

여러 테이블을 조인하여 읽고, 그 데이터의 상태가 일치해야 하는 경우

```
@Transactional(readOnly = true) public List<User> getAllUsers() { r
eturn userRepository.findAll(); }
```

`readOnly = true` 의 의미

Dirty Checking(**변경 감지**) 비활성화 → 성능 최적화

트랜잭션은 유지되지만, **쓰기 작업이 불가능**

JPA, Hibernate 환경에서 읽기 트랜잭션 최적화에 자주 사용