동시성 문제 및 분산 락 - 1

PART1 트랜잭션 및 읽기 일관성

트랜잭션

: 시스템에서 한번의 처리로 실행 되어야 할 나눌 수 없는 업무처리 단위

DB 관점에서 트랜잭션

1. 트랜잭션은 논리적으로 한번에 수행되어야 하는 작업들의 묶음
2. 여러 개의 SQL로 구성됨
3. SQL 작업 실패 시, Rollback 해서 데이터의 일관성을 보장해야함
4. 일반적으로 트랜잭션은 DML, DCL, DDL의 명령어를 하나이상 포함
5. DBMS는 동시에 실행되는 트랜잭션 간의 충돌을 방지하고,
데이터 일관성을 유지하기 위해서 동시성 제어 기법을 사용함

읽기 일관성

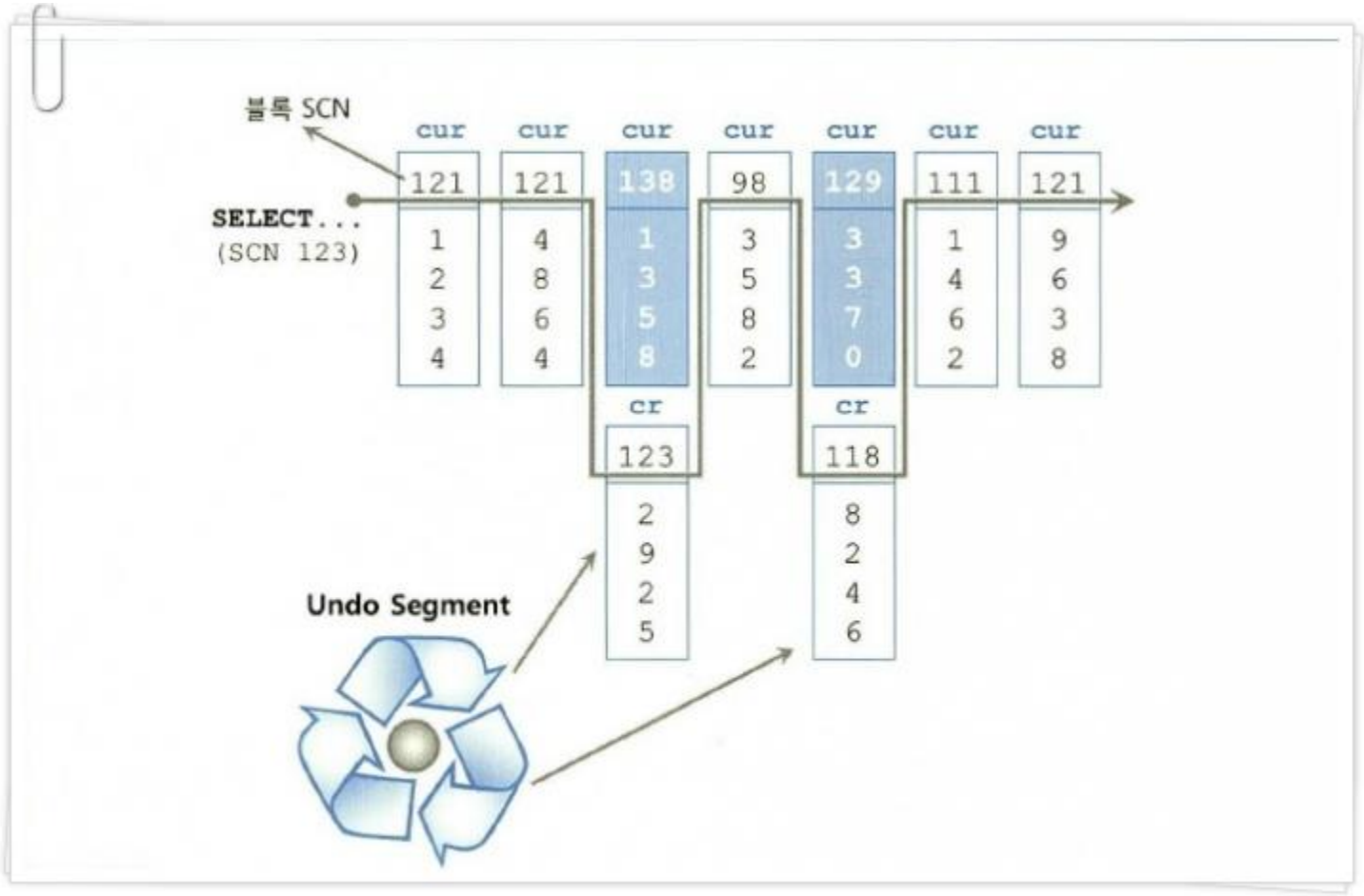
- 문장 수준의 읽기 일관성 : SQL 수준
- 트랜잭션 수준 읽기 일관성 : 트랜잭션 수준

문장 수준 읽기 일관성

Consistent Mode	Current Mode
쿼리가 시작된 시점을 기준으로 Commit된 데이터만 읽기	데이터를 찾아간 현재 시점의 데이터 읽기

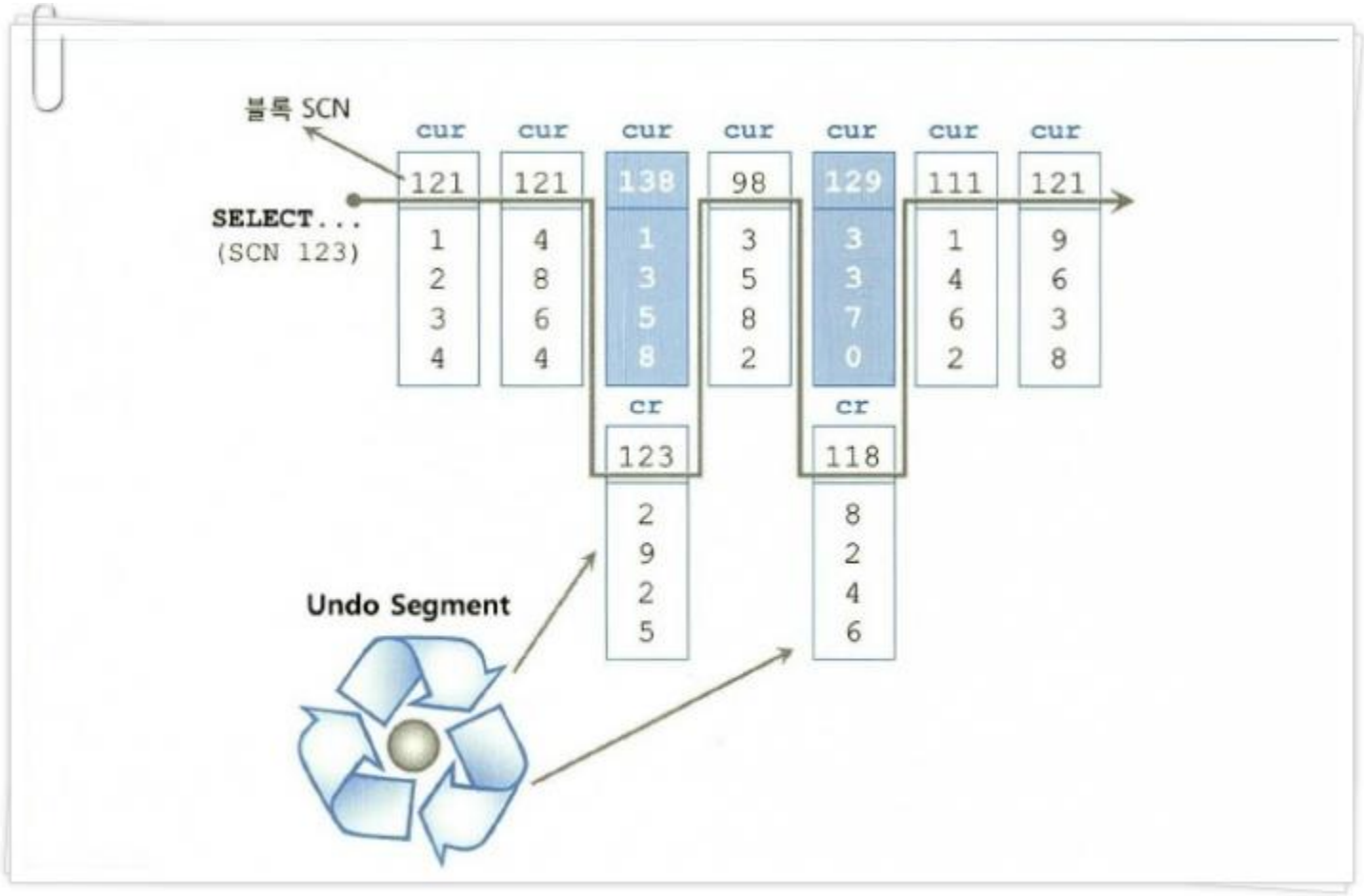
Current 모드 읽기

: **현재 읽는 시점을 기준**으로 Commit된 데이터를 읽는다



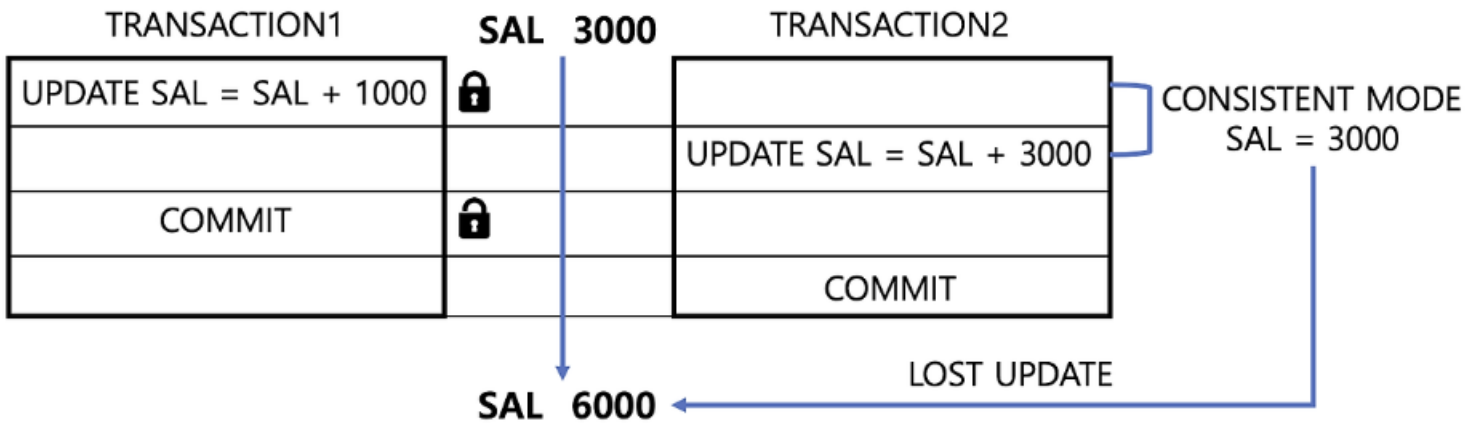
Consistent 모드 읽기

: **Query가 시작된 시점**을 기준으로 Commit된 데이터를 읽는다



Consistent 모드 갱신

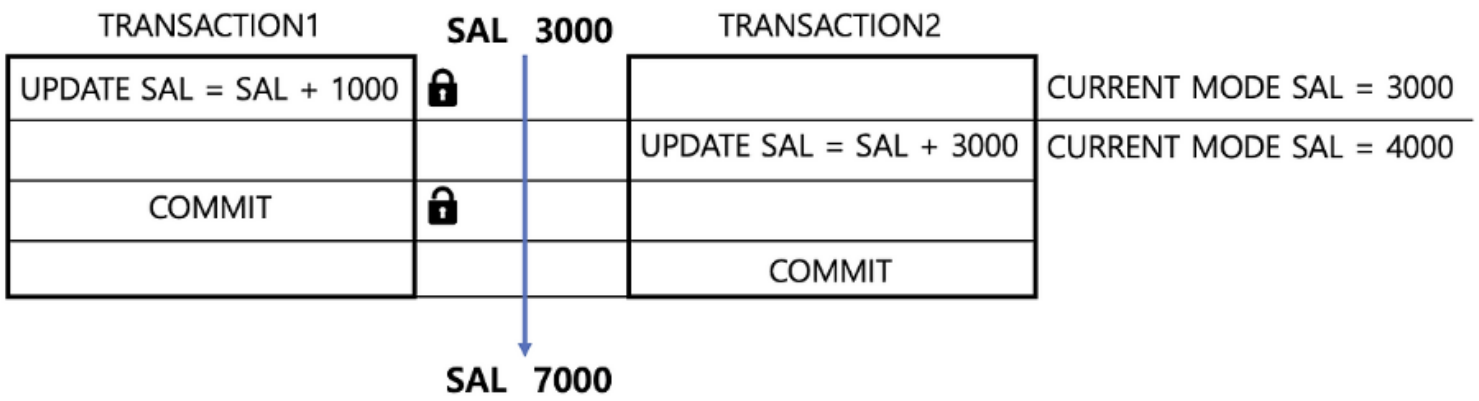
Consistent Mode Update 문제점



Transaction1 Update 시점 SAL = 3000
Transaction2 Update 시점 SAL = 3000

Transaction2에서 Update하는 시점에서
Transaction1이 Commit되지 않았기 때문에 SAL을 아직 3000으로 보고 있음

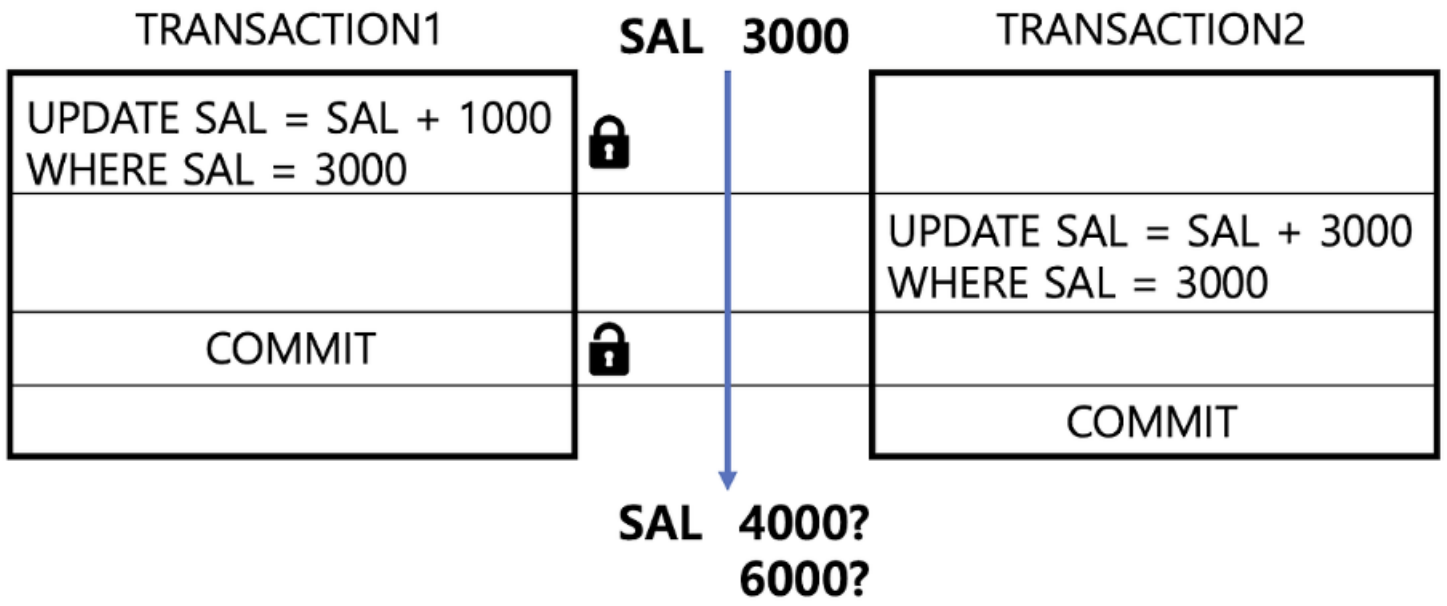
Current 모드 갱신



Transaction1 Update 시점 SAL = 3000
Transaction2 Update 시점 SAL = 4000

현재 데이터를 읽는 시점을 기준으로 하기 때문에 SAL을 4000으로 봄

Consistent 모드 읽기, Current 모드 갱신



Consistent Mode 로 대상을 식별하고, **Current Mode** 로 갱신한다

현실적 제약

- 데이터베이스 시스템은 사용자 함수의 내부 동작을 완전히 제어할 수 없음
- 트랜잭션 내에서도 함수 호출 간 일관성 보장 어려움

재고 관리 예시

```
SELECT
    p.product_id,
    GET_AVAILABLE_STOCK(p.product_id) AS first_check,
    -- 다른 로직...
    CASE WHEN GET_AVAILABLE_STOCK(p.product_id) >
0 THEN '재고 있음'
          ELSE '품절' END AS status
FROM products p;
```

첫 번째 GET_AVAILABLE_STOCK 호출: 재고 10개 반환
잠시 후 다른 세션에서 8개 주문이 들어옴
두 번째 GET_AVAILABLE_STOCK 호출: 재고 2개 반환

결과적으로 같은 SQL 문장 내에서 데이터 불일치가 발생하여 문장 수준 읽기 일관성이 깨집니다.

단일 쿼리로 통합하기

```
-- 함수를 사용하지 않고 로직을 인라인으로 작성
SELECT
    p.product_id,
    p.stock_quantity - IFNULL(SUM(oi.quantity), 0) AS
available_stock,
    CASE WHEN p.stock_quantity -
IFNULL(SUM(oi.quantity), 0) > 0
        THEN '재고 있음' ELSE '품절' END AS status
FROM
    products p
LEFT JOIN
    order_items oi ON p.product_id = oi.product_id AND
oi.status = 'processing'
GROUP BY
    p.product_id, p.stock_quantity;
```

트랜잭션과 락 활용하기

```
START TRANSACTION;

-- 읽기 시점에 락 획득하여 다른 트랜잭션이 변경하지 못
하게 함
SELECT product_id, stock_quantity
FROM products
WHERE product_id = 123
FOR UPDATE;

-- 조건에 맞으면 업데이트
UPDATE products SET stock_quantity = stock_quantity -
5 WHERE product_id = 123;

COMMIT;
```

배치 처리로 일관성 유지하기

```
-- 처리해야 할 주문을 일괄적으로 처리
START TRANSACTION;

-- 처리할 주문 목록 일괄 조회
SELECT order_id, product_id, quantity
FROM orders
WHERE status = 'pending'
LIMIT 100
FOR UPDATE;

-- 재고 현황 일괄 조회
SELECT product_id, stock_quantity
FROM products
WHERE product_id IN (주문된_상품_ID_목록)
FOR UPDATE;

-- 조건 확인 후 재고 일괄 업데이트
UPDATE products
SET stock_quantity = CASE
    WHEN product_id = 101 THEN stock_quantity - 3
    WHEN product_id = 102 THEN stock_quantity - 5
    -- 기타 상품들...
END
WHERE product_id IN (101, 102, ...);

-- 주문 상태 업데이트
UPDATE orders SET status = 'processed' WHERE order_id IN (처리된_주문_ID_목록);

COMMIT;
```