Chapter 08

트랜잭션, 동시성 제어, 회복

MySQL로 배우는 데이터베이스 개론과 실습

목차

학습목표

- 트랜잭션의 개념을 이해하고 데이터베이스에서 이 개념이 왜 중요한지 알아본다.
- 트랜잭션 실행 시 동시성 제어가 필요한 이유를 알아보고, 락킹을 이용한 동시성 제어 기법을 살펴본다.
- 락킹보다 완화된 방법으로 트랜잭션의 동시성을 높이는 트랜잭션 고립 수준에 대해 알아본다.
- 데이터베이스 시스템에 문제가 생겼을 때 쓸 수 있는 복구 방법을 알아본다.

01 트랜잭션

1. 트랜잭션의 개념
2. 트랜잭션의 성질
3. 트랜잭션과 DBMS

1. 트랜잭션의 개념

❖ 트랜잭션

- DBMS에서 데이터를 다루는 논리적인 작업의 단위

❖ 트랜잭션을 정의하는 이유

- 데이터베이스에서 데이터를 다룰 때 장애가 일어나는 경우가 있음. 트랜잭션은 장애 시 데이터를 복구하는 작업의 단위가 된다.
- 데이터베이스에서 여러 작업이 동시에 같은 데이터를 다룰 때가 있음. 트랜잭션은 이 작업을 서로 분리하는 단위가 됨

- 예) A 계좌(박지성)에서 B 계좌(김연아)로 10,000원을 이체할 경우 프로그램

```
① A 계좌(박지성)에서 10,000원을 인출하는 SQL UPDATE 문
② B 계좌(김연아)에   10,000원을 입금하는 SQL UPDATE 문
```

1. 트랜잭션의 개념

❖ 트랜잭션에서 문제가 발생할 수 있는 경우와 해결

- 만약 ①번 SQL 문이 수행된 다음 시스템에 문제가 생기거나 다른 UPDATE 문이 끼어들어 A 계좌에서 돈을 동시에 인출하면, A 계좌와 B 계좌의 잔액이 의도하지 않은 값이 될 수 있음
- ①번 SQL 문과 ②번 SQL 문은 모두 수행되거나 아예 수행되지 않아야 함
- 만약 ①번 SQL 문을 수행한 다음 문제가 생겨 ②번 SQL 문을 수행할 수 없는 경우라면 ①번 SQL 문의 수행을 취소해야 함

- DBMS에 ①번과 ②번의 SQL 문이 하나의 수행 단위라는 것을 알리기 위해 START TRANSACTION 문과 COMMIT 문을 사용하여 트랜잭션의 시작과 끝을 표시함

```
START TRANSACTION;
      ① A 계좌(박지성)에서 10,000원을 인출하는 SQL UPDATE 문
      ② B 계좌(김연아)에 10,000원을 입금하는 SQL UPDATE 문
COMMIT;
```

1. 트랜잭션의 개념

- 계좌이체 트랜잭션이 데이터베이스에서 실행될 때 일어나는 상황을 구체적으로 보면 [그림 8-1]과 같음

테이블: Customer(name, balance)

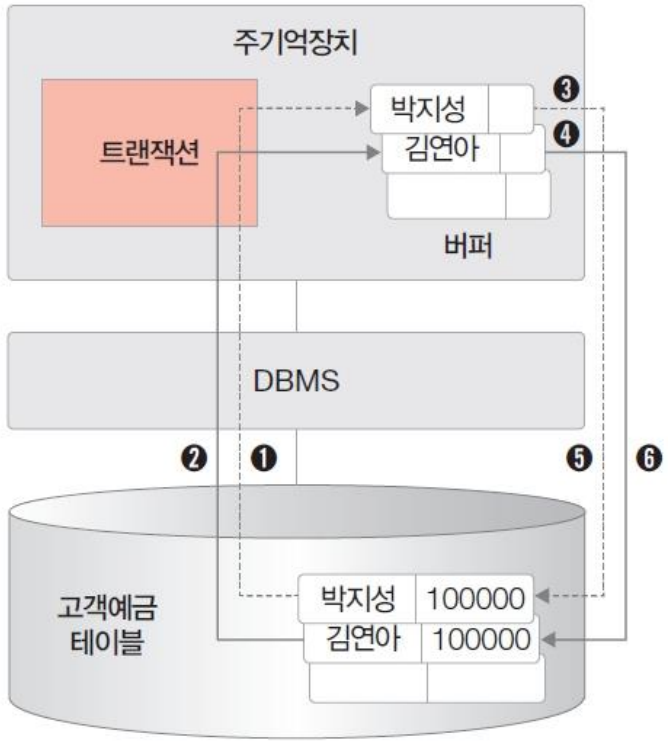
```
START TRANSACTION;
❶ /* 박지성 계좌를 읽어 온다 */
❷ /* 김연아 계좌를 읽어 온다 */
   /* 잔고 확인 */

❸ /* 예금인출 박지성 */
UPDATE   Customer
SET      balance=balance-10000
WHERE    name='박지성';

❹ /* 예금입금 김연아 */
UPDATE   Customer
SET      balance=balance+10000
WHERE    name='김연아';

COMMIT; /* 부분 완료 */
❺ /* 박지성 계좌를 기록한다 */
❻ /* 김연아 계좌를 기록한다 */
```

(a) 계좌이체 트랜잭션



(b) 트랜잭션 수행 과정

그림 8-1 계좌이체 트랜잭션과 수행 과정

1. 트랜잭션의 개념

- COMMIT 문은 트랜잭션의 종료를 알리는 SQL 문
- 계좌이체 트랜잭션의 경우, ❶ ~ ❻이 완전히 끝나야 트랜잭션이 종료되지만, DBMS는 ❶ ~ ❹까지 수행하고 트랜잭션의 수행 내용이 데이터베이스에 완전히 반영되지 않은 상태에서도 부분적인 완료를 함
- 나머지 ❺ ~ ❻은 DBMS가 책임지고 수행함

  [방법 1] ❶ - ❷ - ❸ - ❹ - COMMIT - ❺ - ❻
- 만약 ❺, ❻을 수행한 다음 COMMIT을 수행하면 [방법 2]와 같은 구조가 됨

  [방법 2] ❶ - ❷ - ❸ - ❹ - ❺ - ❻ - COMMIT

- DBMS는 [방법 1]을 택함. [방법 1]을 풀어 쓰면 다음과 같으며, 이는 트랜잭션의 실제 수행 과정과 같음

  [방법 1] ❶ - ❷ - ❸ - ❹ - COMMIT(부분 완료) - ❺ - ❻ - COMMIT(완료)
- 트랜잭션의 수행 과정을 정리하면 [그림 8-2]

그림 8-2 트랜잭션의 수행 과정

2. 트랜잭션의 성질

표 8-1 트랜잭션과 프로그램의 차이점

구분	트랜잭션	프로그램

프로그램 구조	START TRANSACTION;
	…
	COMMIT;
		main( )
		{
		…
		}

다루는 데이터	데이터베이스에 저장된 데이터	파일에 저장된 데이터
번역기	DBMS	컴파일러
특성	원자성, 일관성, 고립성, 지속성	

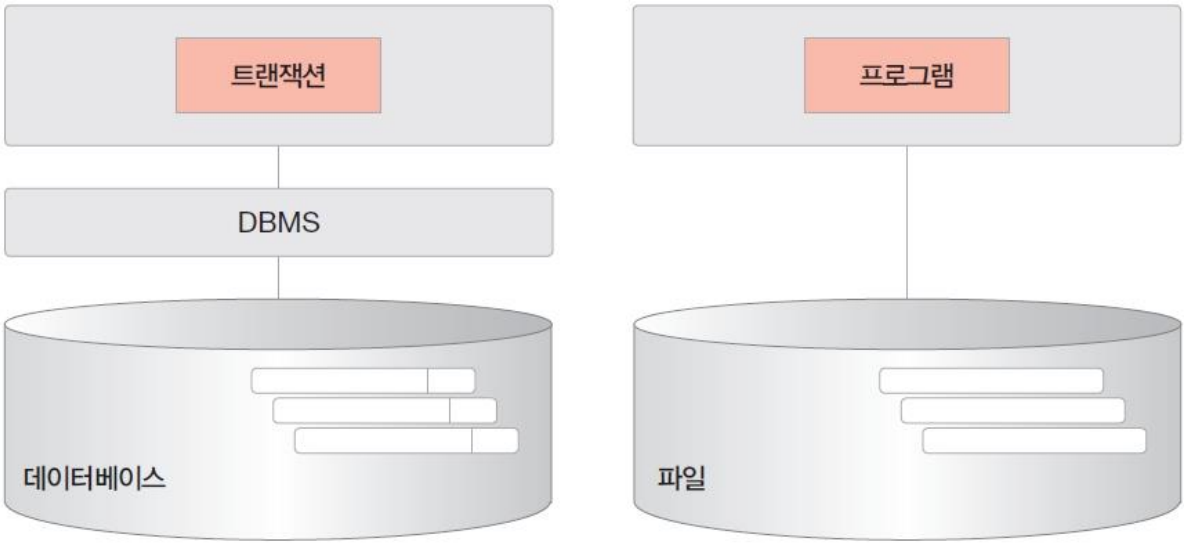


그림 8-3 트랜잭션과 프로그램의 차이점

2. 트랜잭션의 성질

❖ 트랜잭션의 4 가지 성질(ACID 성질)

- **원자성(Atomicity)** : 트랜잭션에 포함된 작업은 전부 수행되거나 아니면 전부 수행되지 않아야 (all or nothing) 함
- **일관성(Consistency)** : 트랜잭션을 수행하기 전이나 수행한 후나 데이터베이스는 항상 일관된 상태를 유지해야 함
- **고립성(Isolation)** : 수행 중인 트랜잭션에 다른 트랜잭션이 끼어들어 변경 중인 데이터 값을 훼손하는 일이 없어야 함
- **지속성(Durability)** : 수행을 성공적으로 완료한 트랜잭션은 변경한 데이터를 영구히 저장해야 함

2. 트랜잭션의 성질

❖ 원자성

- 트랜잭션이 원자처럼 더 이상 쪼개지지 않는 하나의 프로그램 단위로 동작해야 한다는 의미
- 일부만 수행되는 일이 없도록 전부 수행하거나 아예 수행하지 않아야함 (all or nothing)
- COMMIT과 ROLLBACK 명령어를 트랜잭션 제어 명령어(TCL)라고 함

표 8-2 MySQL의 트랜잭션 제어 명령어(TCL)

표준 명령어	문법	설명
START TRANSACTION	SET TRANSACTION	트랜잭션의 시작
COMMIT	COMMIT	트랜잭션의 종료
ROLLBACK	ROLLBACK {TO 〈savepoint〉}	트랜잭션을 전체 혹은 〈savepoint〉까지 무효화시킴
SAVE	SAVEPOINT 〈identifier〉	〈savepoint〉를 만듦

2. 트랜잭션의 성질

- MySQL에서 트랜잭션을 선언하고 수행하는 예

파일명: 8_code01.sql

```
START TRANSACTION;
INSERT INTO Book VALUES(99, '데이터베이스', '한빛', 25000);

SELECT   bookname 'bookname1' FROM Book
     WHERE   bookid=99; /* 데이터베이스 */
SAVEPOINT a;

UPDATE   Book SET bookname='데이터베이스 개론' WHERE bookid=99;
SELECT   bookname 'bookname2' FROM Book
     WHERE   bookid=99; /* 데이터베이스 개론 */
SAVEPOINT b;

UPDATE   Book SET bookname='데이터베이스 개론 및 실습' WHERE bookid=99;
SELECT   bookname 'bookname3' FROM Book
     WHERE   bookid=99; /* 데이터베이스 개론 및 실습 */
ROLLBACK TO b; -- SAVEPOINT b까지 올라가면서 작업을 무효화한다.

SELECT   bookname 'bookname4' FROM Book
     WHERE   bookid=99; /* 데이터베이스 개론 */

ROLLBACK TO a; -- SAVEPOINT a까지 올라가면서 작업을 무효화한다.
```

2. 트랜잭션의 성질

```
SELECT   bookname 'bookname5' FROM Book
     WHERE   bookid=99; /* 데이터베이스 */
COMMIT;

START TRANSACTION; -- 새로운 트랜잭션을 시작한다.
UPDATE   Book SET bookname='데이터베이스 개론 및 실습2' WHERE bookid=99;
SELECT   bookname 'bookname6' FROM Book
     WHERE   bookid=99; /* 데이터베이스 개론 및 실습2 */
ROLLBACK; -- START TRANSACTION 문까지 올라가면서 작업을 무효화한다.

SELECT   bookname 'bookname7' FROM Book
     WHERE   bookid=99; /* 데이터베이스 */

DELETE FROM Book WHERE bookid=99;
COMMIT;
```

2. 트랜잭션의 성질

❖ 일관성

- 테이블이 생성될 때 CREATE 문과 ALTER 문의 무결성 제약조건을 통해 명시됨
- 예) 계좌이체 트랜잭션의 일관성 조건은 'A 계좌 + B 계좌 = 20만 원'

  그런데 A 계좌에서 1만 원을 인출하여 B 계좌에 입금하기 전에는 총액이 일시적으로 19만 원으로 줄어드는 '일관성 없는 inconsistent' 상태가 됨. 트랜잭션이 종료된 후에는 'A 계좌 + B 계좌 = 20만 원'이 됨

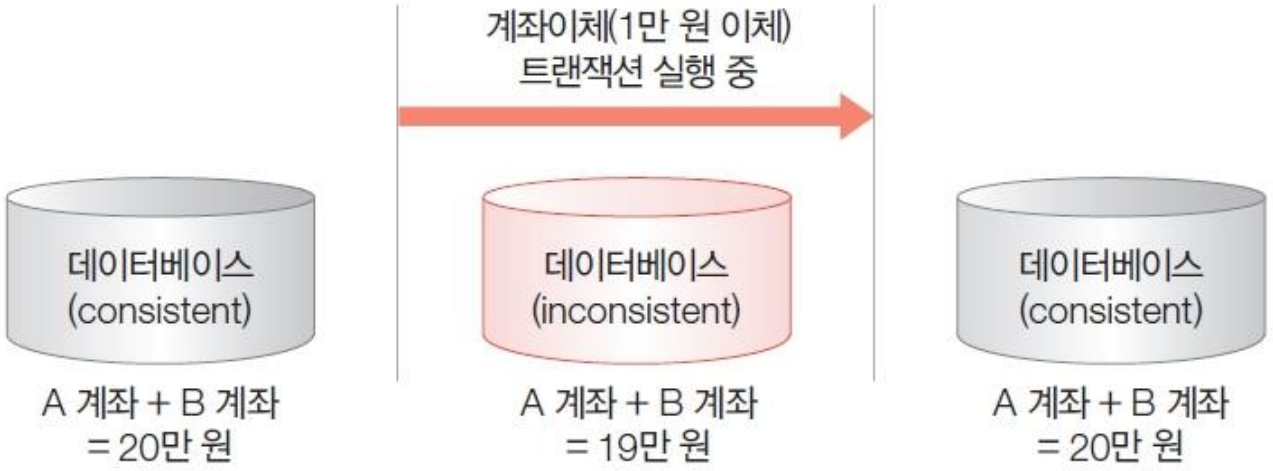


그림 8-4 데이터베이스 변경 중과 변경 후의 일관성

2. 트랜잭션의 성질

❖ 고립성

- 여러 트랜잭션이 동시에 수행될 때 상호 간섭이나 데이터 충돌이 일어나지 않는 현상
- 고립성을 유지하려면 변경 중인 임시 데이터를 다른 트랜잭션이 읽거나 쓰려고 할 때 제어하는 작업이 필요
- 예) 같은 시간대에 여러 트랜잭션이 같은 데이터를 다루고 있음
  - 같은 데이터를 동시에 읽고 쓸 경우, 변경 중인 데이터를 다른 트랜잭션이 사용하면 데이터의 일관성이 훼손될 수 있어 서로 충돌하지 않도록 제어하는 작업이 필요 = 동시성 제어

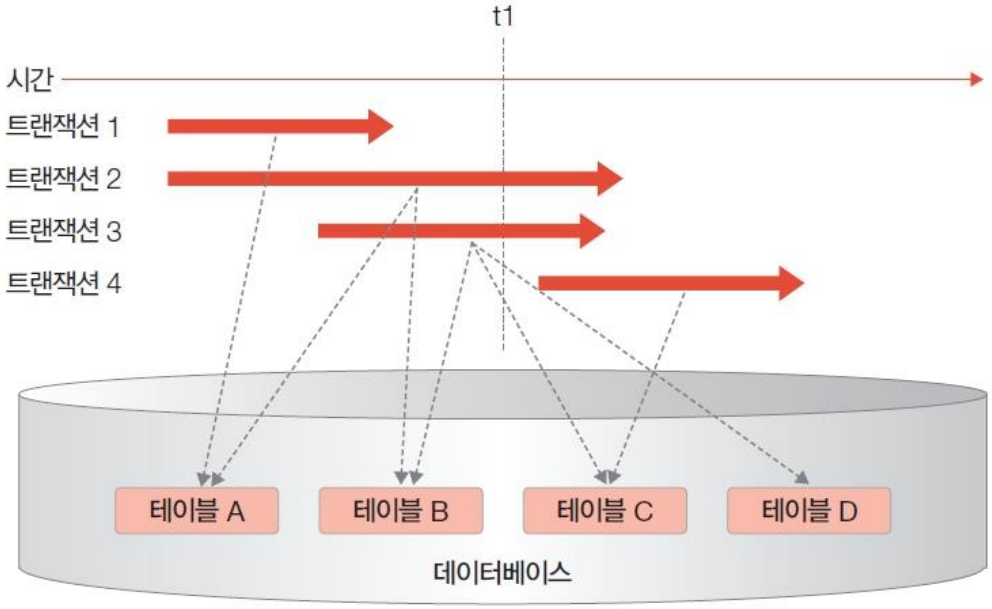


그림 8-5 트랜잭션의 동시 수행과 데이터 공유

2. 트랜잭션의 성질

❖ 지속성

- 트랜잭션이 정상적으로 완료되거나 부분 완료된 데이터는 반드시 데이터베이스에 기록되어야 함
- 트랜잭션의 상태도

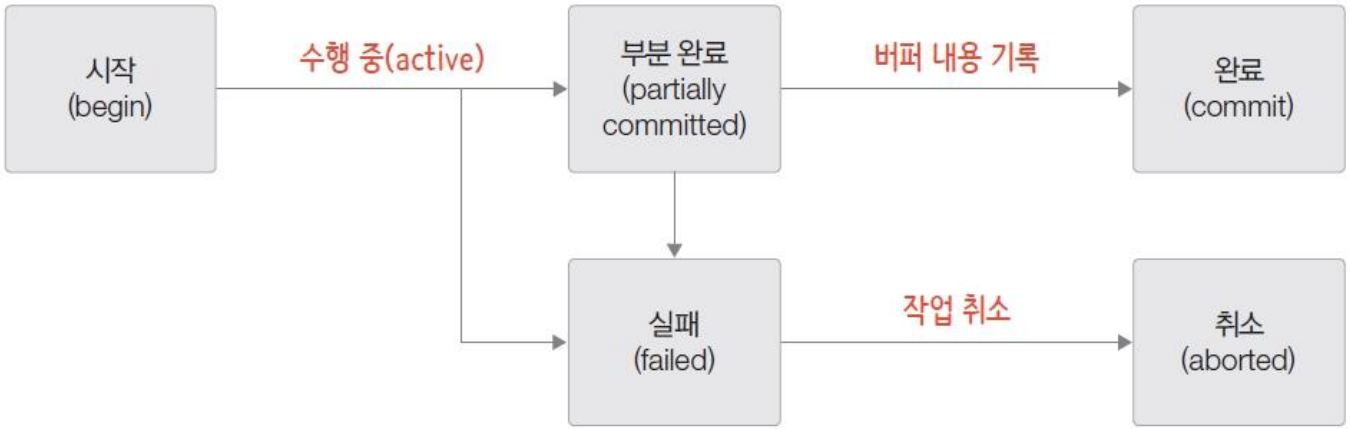


그림 8-6 트랜잭션의 상태도

- 트랜잭션이 수행을 완료하면 부분 완료 혹은 실패 상태 중 하나가 됨
- DBMS는 부분 완료 상태에서는 작업한 내용을 데이터베이스에 반영하고, 실패 상태에서는 작업한 내용을 취소함

3. 트랜잭션과 DBMS

- DBMS는 트랜잭션이 원자성, 일관성, 고립성, 지속성을 유지할 수 있도록 지원함
- 원자성과 지속성을 유지하기 위해 회복(복구) 관리자 프로그램을 작동시킴
- DBMS는 일관성을 유지하기 위해 무결성 제약조건을 활용함
- DBMS는 고립성을 유지하기 위해 일관성을 유지할 때와 마찬가지로 동시성 제어 알고리즘을 작동시킴

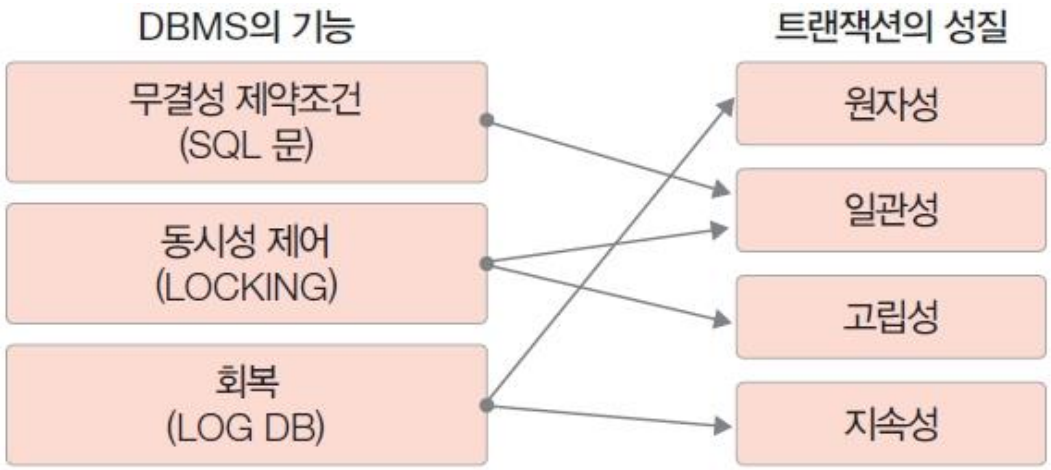


그림 8-7 DBMS의 기능과 트랜잭션의 성질

연습문제 (Q8.1)

01 트랜잭션은 일련의 연산 집합이란 의미로, 하나의 논리적인 기능을 수행하는 작업의 단위다. 트랜잭션이 가져야 할 성질로 거리가 먼 것은?

① 원자성(Atomicity)
② 고립성(Isolation)
③ 지속성(Durability)
④ 동시성(Concurrency)

02 트랜잭션에 대한 설명으로 옳지 않은 것은?

① 한꺼번에 모두 수행되어야 할 일련의 데이터베이스 연산 집합이다.
② 데이터베이스의 일관성을 유지하는 프로그램 단위다.
③ 동시성 제어 및 회복의 논리적인 작업 단위다.
④ 트랜잭션 연산이 데이터베이스에 모두 반영되지 않고 일부만 반영되는 것을 원자성이라고 한다.

03 트랜잭션의 성질에 대한 설명으로 옳지 않은 것은?

① 트랜잭션의 연산은 데이터베이스에 모두 반영되든지, 아니면 모두 반영되지 않아야 한다.
② 트랜잭션 수행이 성공적으로 완료되면 데이터베이스는 언제나 일관성 있는 상태를 유지해야 한다.
③ 어느 하나의 트랜잭션 실행 중에 다른 트랜잭션이 동시에 실행될 수 없다.
④ 트랜잭션에 의해서 생성된 결과는 데이터베이스에 영구히 저장되어야 한다.

02 동시성 제어

1. 동시성 제어의 개념
2. 갱신 손실 문제
3. 락

1. 동시성 제어의 개념

- 한 개의 트랜잭션을 끝내고 다음 트랜잭션을 수행시키면 데이터베이스의 일관성에 문제가 없음
- 그러나 데이터베이스는 공유를 목적으로 하므로 가능한 한 많은 트랜잭션을 동시에 수행시켜야 함
- 트랜잭션이 동시에 수행될 때, 일관성을 해치지 않도록 트랜잭션의 데이터 접근을 제어하는 DBMS의 기능을 동시성 제어라고 함

- [표 8-3]은 두 개의 트랜잭션이 한 개의 데이터에 동시 접근할 때 발생할 수 있는 상황을 정리한 것

표 8-3 트랜잭션의 읽기/쓰기 시나리오

상황	트랜잭션 1	트랜잭션 2	발생 문제	동시 접근
상황 1	읽기	읽기	없음(읽기만 하면 아무 문제가 없음)	허용
상황 2	읽기	쓰기	오손 읽기, 반복불가능 읽기, 유령데이터 읽기	허용/불가 중 선택
상황 3	쓰기	쓰기	갱신 손실(절대 허용하면 안 됨)	허용 불가(LOCK 이용)

2. 갱신 손실 문제

- **갱신손실(lost update)** : 두 개의 트랜잭션이 한 개의 데이터를 동시에 갱신할 때 발생함, 갱신 손실 문제는 데이터베이스에서 절대로 발생하면 안 되는 현상

트랜잭션 T1, T2가 있다. T1은 예금을 인출(UPDATE)하는 작업을 하고, T2는 입금(UPDATE)하는 작업을 한다. T1은 계좌 X에서 100을 빼고, T2는 계좌 X에 100을 더한다. 초기에 X의 값이 1000이라면 T1 → T2 혹은 T2 → T1 어느 순서로 실행해도 결과는 X = 1000이 되어야 한다. 이 경우 일관성 조건은 X 계좌의 총합은 1000이라는 것이다.

트랜잭션 T1	트랜잭션 T2	버퍼의 데이터 값
A = read_item(X); ① A = A-100;		X = 1000
	B = read_item(X); ② B = B+100;	X = 1000
write_item(A → X); ③		X = 900
	write_item(B → X); ④	X = 1100

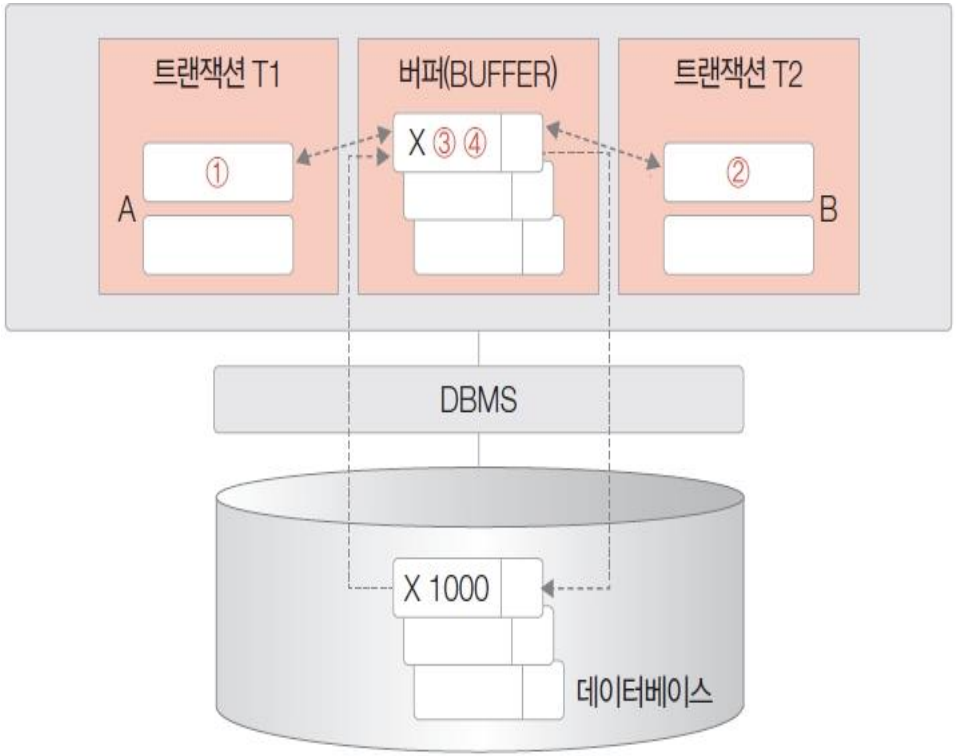


그림 8-8 갱신 손실 문제 발생 시나리오

3. 락

❖ 락(lock)의 개념

- 트랜잭션이 데이터를 읽거나 수정할 때 데이터에 표시하는 잠금장치
- 락을 사용해 [그림 8-8]의 예를 수정하면 [그림 8-9]와 같음

트랜잭션 T1	트랜잭션 T2	버퍼의 데이터 값
LOCK(X) A = read_item(X); ① A = A−100;		X = 1000
	LOCK(X) (wait… 대기)	X = 1000
write_item(A→X); ② UNLOCK(X);		X = 900
	B = read_item(X); ③ B = B+100; write_item(B→X); ④ UNLOCK(X)	X = 1000

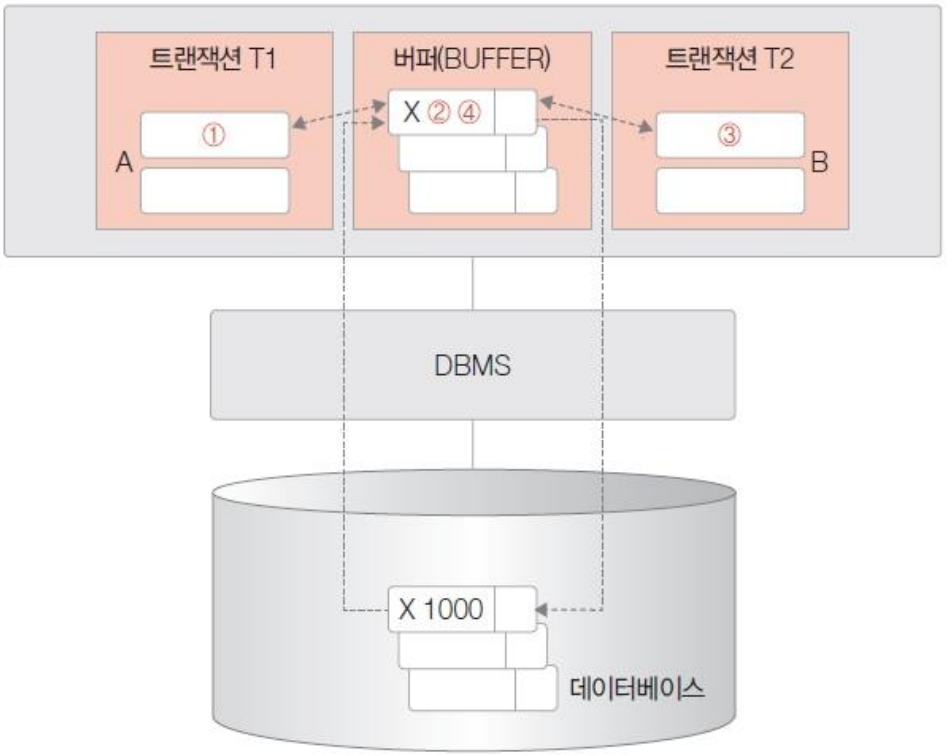


그림 8-9 락을 이용한 갱신 손실 문제 해결 시나리오

3. 락

- 하나의 데이터에 두 트랜잭션이 접근하여 갱신하는 작업의 예로 각 트랜잭션의 SQL 문은 [표 8-4]와 같음

트랜잭션 T1, T2는 같은 데이터를 수정한다. T1은 1번 도서의 가격을 조회한 후 7,100원으로 수정하고, T2는 1번 도서의 가격을 조회한 후 가격을 100원 인상한다.

표 8-4 하나의 데이터에 두 트랜잭션이 접근하여 갱신하는 작업 예

트랜잭션 T1	트랜잭션 T2
START TRANSACTION; USE madangdb; SELECT * FROM Book WHERE bookid=1; UPDATE Book SET price=7100 WHERE bookid=1; SELECT * FROM Book WHERE bookid=1; COMMIT;	START TRANSACTION; USE madangdb; SELECT * FROM Book WHERE bookid=1; UPDATE Book SET price=price+100 WHERE bookid=1; SELECT * FROM Book WHERE bookid=1; COMMIT;

3. 락

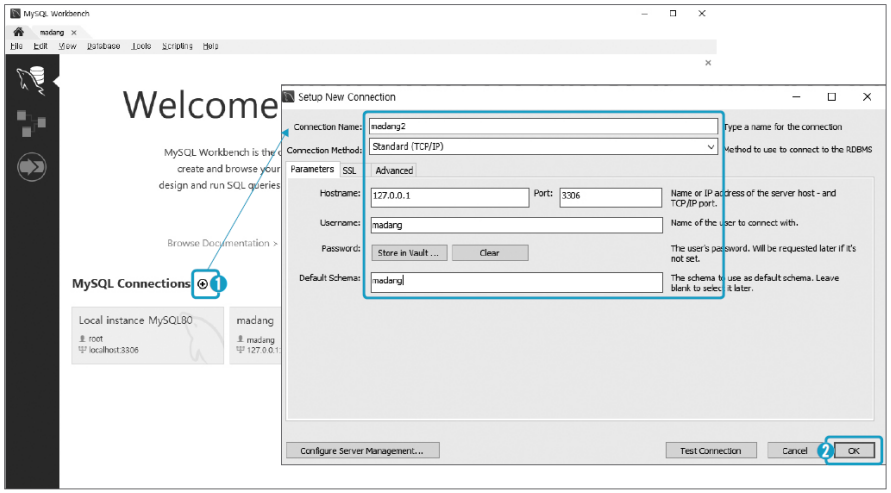
❖여기서 잠깐 : MySQL에서 두 트랜잭션을 동시에 실행시키는 방법

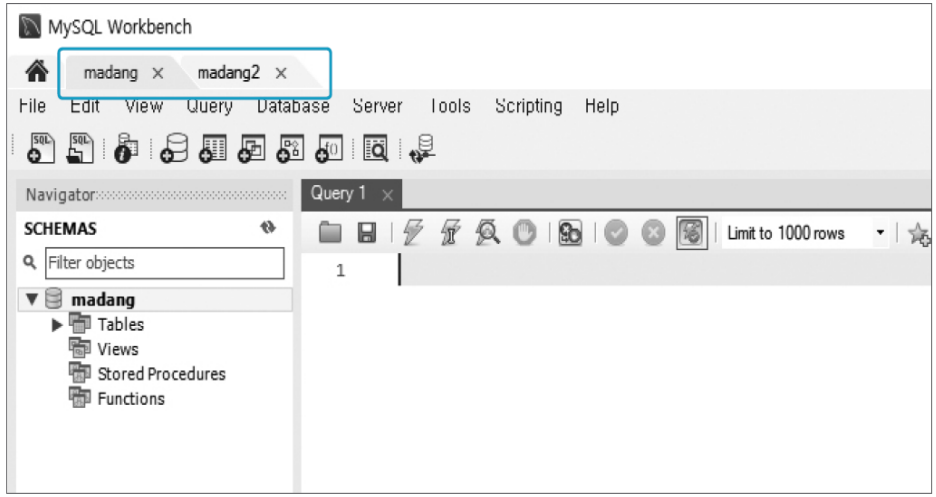
■ 트랜잭션 실습을 진행하기 위해서는 **MySQL** 접속 시 서로 다른 두 세션(세션은 데이터베이스 접속단위)에서 진행해야 한다.

■ 방법

❶ 두 번째 세션을 위해 **Workbench** 초기화면에서 **madang2** 접속을 만든다.
(madang과 동일한 내용으로 **madang2**를 만듦).

❷ 실습을 위해 **madang, madang2** 접속





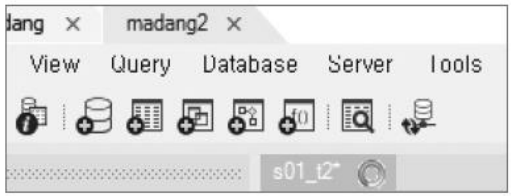
3. 락

- [표 8-4]의 두 트랜잭션이 동시에 실행되는 상황 (파일명: s01_t1.sql, s01_t2.sql)

표 8-5 [표 8-4]의 두 트랜잭션이 동시에 실행되는 상황(파일명: s01_t1.sql, s01_t2.sql)

트랜잭션 T1	트랜잭션 T2
START TRANSACTION; USE madangdb; SELECT * FROM Book WHERE bookid=1; bookid | bookname | publisher | price 1 | 축구의 역사 | 굿스포츠 | 7000 UPDATE Book SET price=7100 WHERE bookid=1;	
	START TRANSACTION; USE madangdb; SELECT * FROM Book WHERE bookid=1; bookid | bookname | publisher | price 1 | 축구의 역사 | 굿스포츠 | 7000 UPDATE Book SET price=price+100 WHERE bookid=1;

3. 락

트랜잭션 T1	트랜잭션 T2
SELECT * FROM Book WHERE bookid=1; bookid | bookname | publisher | price 1 | 축구의 역사 | 굿스포츠 | 7100 COMMIT;	…(30초간 대기)… 
	SELECT * FROM Book WHERE bookid=1; bookid | bookname | publisher | price 1 | 축구의 역사 | 굿스포츠 | 7200 COMMIT;

3. 락

❖ 락의 유형

- **공유락(LS, shared lock)** : 트랜잭션이 읽기를 할 때 사용하는 락
- **배타락(LX, exclusive lock)** : 읽고 쓰기를 할 때 사용하는 락

❖ 트랜잭션이 데이터 X에 대한 공유락과 배타락을 사용하는 규칙

- 데이터에 락이 걸려 있지 않으면 트랜잭션은 데이터에 락을 걸 수 있음
- 트랜잭션이 데이터 X를 읽기만 하면 LS(X)를 요청하고, 읽고 쓰기를 하면 LX(X)를 요청함
- 트랜잭션 T1이 데이터에 LS(X)를 걸어두면, 트랜잭션 T2의 LS(X) 요청은 허용하고 LX(X)는 허용하지 않음
- 트랜잭션 T1이 데이터에 LX(X)를 걸어두면, 트랜잭션 T2의 LS(X)와 LX(X) 모두 허용하지 않음
- 트랜잭션이 락을 허용받지 못하면 대기 상태가 됨
- 락 호환행렬(lock compatibility matrix)

표 8-6 락 호환행렬

요청 \ 상태	LS 상태	LX 상태
LS 요청	허용	대기
LX 요청	대기	대기

3. 락

❖ 2단계 락킹(2 phase locking protocol)

락을 걸고 해제하는 시점에 제한을 두지 않으면 두 개의 트랜잭션이 동시에 실행될 때 데이터의 일관성이 깨질 수 있어 이를 방지하는 방법

- 확장단계(Growing phase, Expanding phase) : 트랜잭션이 필요한 락을 획득하는 단계로, 이 단계에서는 이미 획득한 락을 해제하지 않음
- 수축단계(Shrinking phase) : 트랜잭션이 락을 해제하는 단계로, 이 단계에서는 새로운 락을 획득하지 않음.

- 표8-7 예제 설명

트랜잭션 T1, T2가 있다. T1은 예금 이체 작업을 하고 T2는 이자 계산 작업을 한다. 초기 A, B 두 데이터 값은 1000이다. 각 트랜잭션의 세부 작업을 보면 T1은 A 계좌에서 100을 인출하여 B 계좌에 입금하고, T2는 A, B 두 계좌의 잔고를 10%씩 증가시킨다. 작업 결과는 T1 → T2 혹은 T2 → T1 어느 순서로 실행해도 A+B=2200이 되어야 한다. 그러나 순서에 차이는 있을 수 있다. T1 → T2의 결과는 A=(1000−100)×1.1=990, B=(1000+100)×1.1=1210으로 A+B=2200이 된다. T2 → T1의 결과는 A=1000×1.1−100=1000, B=1000×1.1+100=1200으로 A+B=2200이 된다. 여기서 암시적인 데이터 일관성 제약조건은 두 개의 트랜잭션 수행 결과 'A+B=2200이 되어야 한다'는 것이다.

3. 락

- [표 8-7]은 T1과 T2를 동시에 실행했을 때 락을 사용했는데도 일관성이 깨지는 이상한 결과가 나오는 예

표 8-7 두 개의 트랜잭션이 접근하여 갱신하는 작업에 락을 사용한 예

트랜잭션 T1	트랜잭션 T2	A, B 값
LX(A) t1 = read_item(A); t1 = t1−100; A = write_item(t1); UN(A)		A = 900 B = 1000

	LX(A)
	t2 = read_item(A);
	t2 = t2*1.1;
	A = write_item(t2);
	UN(A)
	LX(B)
	t2 = read_item(B);
	t2 = t2*1.1;
	B = write_item(t2);
	UN(B)
		A = 990
		B = 1100

LX(B)
t1 = read_item(B);
t1 = t1+100;
B = write_item(t1);
UN(B)
		A = 990
		B = 1200
		/* A+B = 2190이므로 일관성 제약조건에 위배됨 */

3. 락

- 확장 단계에서는 락을 걸기만 하고 수축단계에서는 락을 해지하기만 하여 일관성이 깨지는 문제를 해결함

표 8-8 두 개의 트랜잭션이 접근하여 갱신하는 작업에 2단계 락을 사용한 예

트랜잭션 T1	트랜잭션 T2	A, B 값
LX(A) t1 = read_item(A); t1 = t1−100; A = write_item(t1);		A = 900 B = 1000
	LX(A) …(대기 상태)…	

LX(B)
t1 = read_item(B);
t1 = t1+100;
B = write_item(t1);
UN(A)
UN(B)
		A = 900
		B = 1100

	LX(A)
	t2 = read_item(A);
	t2 = t2*1.1;
	A = write_item(t2);
	LX(B)
	t2 = read_item(B);
	t2 = t2*1.1;
	B = write_item(t2);
	UN(A)
	UN(B)
		A = 990
		B = 1210
		/* A+B = 22000이므로 일관성 제약조건을 지킴 */

3. 락

❖ 데드락(교착상태, deadlock)

- 두 개 이상의 트랜잭션이 각각 자신의 데이터에 대하여 락을 획득하고 상대방 데이터에 대하여 락을 요청하면 무한 대기 상태에 빠질 수 있음. 이러한 현상을 말함

- 데드락이 어떻게 발생하는지 예제

> 마당서점 데이터에 접근하는 트랜잭션 T1, T2가 있다. T1은 도서번호가 1인 도서와 도서 번호가 2인 도서의 가격을 100원 올리고, T2는 역순으로 도서번호가 2인 도서와 도서번호가 1인 도서의 가격을 10% 인상한다.

  - Workbench에서 T1, T2에 대하여 두 개의 쿼리 창을 만든 후 번갈아 가면서 [표 8-9]와 같은 순서로 실행해 보면 데드락이 발생하는 것을 관찰할 수 있음

3. 락

표 8-9 두 개의 트랜잭션이 접근하여 갱신하려고 하여 데드락이 발생한 상황 예(파일명: c8_s02_t1.sql, c8_s02_t2.sql)

트랜잭션 T1	트랜잭션 T2
START TRANSACTION; USE madangdb; UPDATE Book SET price=price+100 WHERE bookid=1;	
	START TRANSACTION; USE madangdb; UPDATE Book SET price=price*1.1 WHERE bookid=2;
UPDATE Book SET price=price+100 WHERE bookid=2; …(대기상태)…	UPDATE Book SET price=price*1.1 WHERE bookid=1; -- Error Code: 1213. Deadlock found when trying to get lock; try restarting transaction
COMMIT;	COMMIT;

3. 락

- 대기 그래프를 그려보면 데드락의 발생 여부를 판단할 수 있음
- 대기 그래프에서 사이클이 존재하면 데드락이 발생한 것

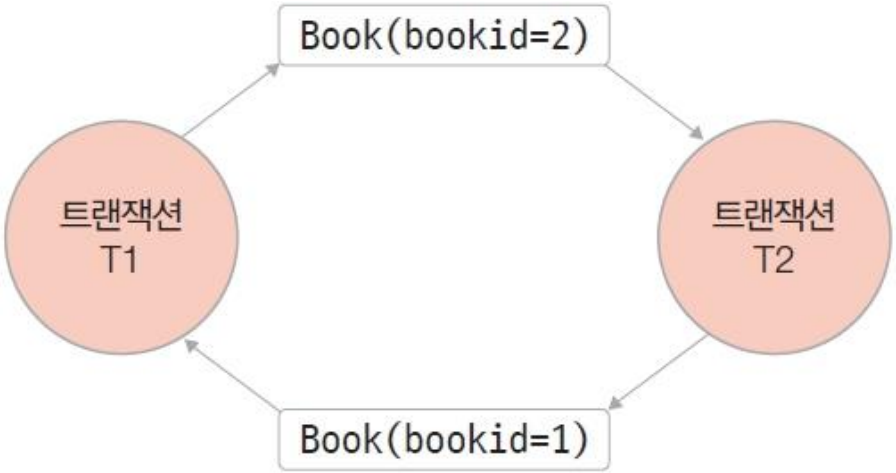


그림 8-10 대기 그래프

연습문제 (Q8.2)

04 트랜잭션의 동시성에 관한 설명 중 옳지 않은 것은?

① 락을 사용하면 갱신 손실 문제를 해결할 수 있다.

② 데드락이 발생하려면 반드시 트랜잭션과 데이터 아이템이 두 개 이상 관여해야 한다.

③ 데이터에 공유락(LS)이 걸려 있다면 다른 트랜잭션의 공유락을 허용해도 된다.

④ 2단계 락킹을 사용하면 데드락 현상은 발생하지 않는다.

05 동시성 제어의 락킹 단위에 대한 설명으로 옳지 않은 것은?

① 락킹 단위가 작아지면 동시성 수준이 낮아진다.

② 데이터베이스, 파일, 레코드 등은 락킹 단위가 될 수 있다.

③ 락킹 단위가 작아지면 락킹 오버헤드가 증가한다.

④ 한꺼번에 락킹할 수 있는 데이터의 크기를 락킹 단위라고 한다.

03 트랜잭션 고립 수준

1. 트랜잭션 동시 실행 문제
2. 트랜잭션 고립 수준 명령어
3. MySQL의 트랜잭션

1. 트랜잭션 동시 실행 문제

- 지금까지 설명한 락은 458쪽 [표 8-3]의 [상황 3]인 두 트랜잭션이 (쓰기, 쓰기)인 상황을 해결하기 위한 것

구분	트랜잭션1	트랜잭션2	발생 문제	동시접근
[상황 1]	읽기	읽기	읽음(읽기만 하면 아무 문제가 없음)	허용
[상황 2]	읽기	쓰기	오손 읽기, 반복불가능 읽기, 유령 데이터 읽기	허용 혹은 불가 선택
[상황 3]	쓰기	쓰기	갱신손실(절대 허용하면 안 됨)	허용불가(LOCK을 이용)

- [표 8-3]의 [상황 2]인 (읽기, 쓰기)
  2개의 트랜잭션에서 트랜잭션 1은 읽기, 트랜잭션 2는 쓰기 작업을 함

✓ [상황 2]에서도 락을 사용하여 두 트랜잭션을 동시에 실행시킬 수 있음

✓ 그러나 [상황 2]는 [상황 3]과 같이 처리하기에는 아쉬운 점이 있음

✓ [상황 2]를 락으로 해결하면 두 트랜잭션의 동시 진행 정도를 과도하게 막기 때문

✓ [상황 2]에서 트랜잭션의 동시성을 높이려면 좀 더 완화된 방법을 사용해야 함

✓ [상황 2]의 트랜잭션 1은 읽기만 하므로 갱신 손실 같은 심각한 문제가 발생하지 않음

✓ 그러나 트랜잭션 1에서 오손 읽기 문제, 반복불가능 읽기 문제, 유령데이터 읽기 문제 등이 발생할 수 있음

✓ 읽기만 하는 트랜잭션이 쓰기 트랜잭션에서 작업한 중간 데이터를 읽기 때문에 발생하는 문제들

1. 트랜잭션 동시 실행 문제

❖ 오손 읽기 (dirty read)

- 읽기 작업을 하는 트랜잭션 1이 쓰기 작업을 하는 트랜잭션 2가 작업한 중간 데이터를 읽기 때문에 생기는 문제

파일명: 8_code_02.sql

```
USE madangdb;
DROP TABLE IF EXISTS Users;

CREATE  TABLE Users
(id     INTEGER,
 name   VARCHAR(20),
 age    INTEGER);

INSERT INTO Users VALUES (1, 'HONG GILDONG', 30);

SELECT *
FROM   Users;
```

id	name	age
1	HONG GILDONG	30

```
COMMIT;
```

1. 트랜잭션 동시 실행 문제

- 다음과 같이 두 트랜잭션이 동시에 실행되는 상황을 가정

> 트랜잭션 T1, T2가 동시에 실행된다. T1은 읽기만 하고 T2는 쓰기를 한다. T1은 T2가 변경한 데이터를 읽어 와 작업하는데, T2가 작업 중 롤백하였다.

표 8-10 오손 읽기 예(파일명: s03_t1.sql, s03_t2.sql)

T1 (읽는 트랜잭션) READ UNCOMMITTED 모드	T2 (쓰는 트랜잭션) READ COMMITTED 모드
SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED; START TRANSACTION; USE madangdb; SELECT * FROM Users WHERE id=1;	

id	name	age
1	HONG GILDONG	30

홍길동의 나이는 30

1. 트랜잭션 동시 실행 문제

T1 (읽는 트랜잭션) READ UNCOMMITTED 모드	T2 (쓰는 트랜잭션) READ COMMITTED 모드
	SET TRANSACTION ISOLATION LEVEL READ COMMITTED; START TRANSACTION; USE madangdb; SET SQL_SAFE_UPDATES=0; UPDATE Users SET age=21 WHERE id=1; SELECT * FROM Users WHERE id=1; id | name | age 1 | HONG GILDONG | 21
SELECT * FROM Users WHERE id=1; id | name | age 1 | HONG GILDONG | 21 홍길동의 나이는 21로 바뀜	
	ROLLBACK;
SELECT * FROM Users WHERE id=1; id | name | age 1 | HONG GILDONG | 30 홍길동의 나이는 30으로 다시 바뀜 COMMIT;	

1. 트랜잭션 동시 실행 문제

❖ 반복불가능 읽기 (non-repeatable read)

- 트랜잭션 1이 데이터를 읽고 트랜잭션 2가 데이터를 쓰고(갱신, UPDATE) 트랜잭션 1이 다시 한번 데이터를 읽을 때 생기는 문제
- 다음과 같이 두 트랜잭션이 동시에 실행되는 상황을 가정

트랜잭션 T1, T2가 동시에 실행된다. T1은 읽기만 하고 T2는 쓰기(갱신, UPDATE)를 한다. T1이 데이터를 읽고 작업을 한 후, T2가 변경한 데이터를 다시 한번 읽어 와 작업한다.

표 8-11 반복불가능 읽기 예(파일명: s04_t1.sql, s04_t2.sql)

T1 (읽는 트랜잭션) READ COMMITTED 모드	T2 (쓰는 트랜잭션) READ COMMITTED 모드
SET TRANSACTION ISOLATION LEVEL READ COMMITTED; START TRANSACTION; USE madangdb; SELECT * FROM Users WHERE id=1;	

id	name	age
1	HONG GILDONG	30

홍길동의 나이는 30

1. 트랜잭션 동시 실행 문제

```
SET TRANSACTION ISOLATION LEVEL READ
COMMITTED;
START TRANSACTION;
USE madangdb;
SET SQL_SAFE_UPDATES=0;
UPDATE  Users
SET     age=21
WHERE   id=1;

COMMIT;

SELECT  *
FROM    Users
WHERE   id=1;
```

id	name	age
1	HONG GILDONG	21

```
SELECT  *
FROM    Users
WHERE   id=1;
```

id	name	age
1	HONG GILDONG	21

홍길동의 나이는 21, 처음과 다른 값

```
COMMIT;

-- 다음 실습을 위해 초기화
UPDATE  Users
SET     age=30
WHERE   id=1;
```

1. 트랜잭션 동시 실행 문제

❖ 유령데이터 읽기 (phantom read)

- 트랜잭션 1이 데이터를 읽고 트랜잭션 2가 데이터를 쓰고(삽입, INSERT) 트랜잭션 1이 다시 한번 데이터를 읽을 때 생기는 문제
- 트랜잭션 1이 읽기 작업을 다시 한번 반복할 경우 이전에 없던 데이터(유령데이터)가 보이는 현상을 유령데이터 읽기라고 함
- 다음과 같이 두 트랜잭션이 동시에 실행되는 상황을 가정

트랜잭션 T1은 읽기만 하고 T2는 쓰기(삽입, INSERT)477를 한다. T1은 데이터를 읽고 작업한 후, 그 사이에 T2가 변경한 데이터를 다시 한번 읽어 와 작업한다.

표 8-12 유령데이터 읽기 예(파일명: s05_t1.sql, s05_t2.sql / MySQL 실습 불가능)

T1 (읽는 트랜잭션) REPEATABLE READ 모드	T2 (쓰는 트랜잭션) READ COMMITTED 모드
SET TRANSACTION ISOLATION LEVEL REPEATABLE READ; START TRANSACTION; USE madangdb; SELECT * FROM Users WHERE age BETWEEN 10 AND 30;	
id | name | age 1 | HONG GILDONG | 30	홍길동의 나이는 30

1. 트랜잭션 동시 실행 문제

T1 (읽는 트랜잭션) REPEATABLE READ 모드	T2 (쓰는 트랜잭션) READ COMMITTED 모드
	SET TRANSACTION ISOLATION LEVEL READ COMMITTED; START TRANSACTION; USE madangdb; INSERT INTO Users VALUES (3, 'Bob', 27); COMMIT; SELECT * FROM Users WHERE age BETWEEN 10 AND 30; id | name | age 1 | HONG GILDONG | 30 3 | Bob | 27
SELECT * FROM Users WHERE age BETWEEN 10 AND 30; id | name | age 1 | HONG GILDONG | 30 3 | Bob | 27 COMMIT; -- 이후 실습을 위해 삽입한 데이터 삭제 DELETE FROM Users WHERE age='27';	

Bob는 없던 데이터

1. 트랜잭션 동시 실행 문제

- MySQL에서는 유령데이터 읽기가 발생하지 않음
- MySQL의 REPEATABLE READ 모드에서는 트랜잭션이 처음 데이터를 읽어 올 때 SNAPSHOT을 구축하여 자료를 가져옴
- 그에 따라 다른 세션의 자료가 변경되더라도 동일한 결과를 보여주게 됨

> REPEATABLE READ (MySQL 8.0 Reference Manual 참조)
>
> This is the default isolation level for InnoDB. Consistent reads within the same transaction read the snapshot established by the first read.

2. 트랜잭션 고립 수준 명령어

- 오손 읽기, 반복불가능 읽기, 유령데이터 읽기의 세 가지 문제를 해결하려면 트랜잭션 수행 시 락을 사용하여 다른 트랜잭션의 간섭을 최소화해야 함
- 하지만 여기서 락을 사용하면 트랜잭션의 동시성이 과하게 제한될 수 있으므로 DBMS는 동시성을 높이는 선택을 할 수 있는 명령어, 즉 트랜잭션 고립 수준 명령어를 제공하여 락을 완화함

표 8-13 트랜잭션 고립 수준 명령어와 발생 현상

문제 / 고립 수준	오손 읽기	반복불가능 읽기	유령데이터 읽기
READ UNCOMMITTED	가능	가능	가능
READ COMMITTED	불가능	가능	가능
REPEATABLE READ	불가능	불가능	가능
SERIALIZABLE	불가능	불가능	불가능

2. 트랜잭션 고립 수준 명령어

❖ READ UNCOMMITTED(Level = 0)

표 8-14 READ UNCOMMITTED 모드 요약

모드	READ UNCOMMITTED
LOCK	SELECT 문 – 공유락 걸지 않음 UPDATE 문 – 배타락 설정함 다른 트랜잭션의 공유락과 배타락이 걸린 데이터를 읽음
SQL 문	SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED
문제점	오손 읽기, 반복불가능 읽기, 유령데이터 읽기

❖ READ COMMITTED(Level = 1)

표 8-15 READ COMMITTED 모드 요약

모드	READ COMMITTED
LOCK	SELECT 문 – 공유락을 걸고 끝나면 바로 해지함 UPDATE 문 – 배타락 설정함 다른 트랜잭션이 설정한 공유락은 읽지만 배타락은 읽지 못함
SQL 문	SET TRANSACTION ISOLATION LEVEL READ COMMITTED
문제점	반복불가능 읽기, 유령데이터 읽기

2. 트랜잭션 고립 수준 명령어

❖ REPEATABLE READ(Level = 2)

표 8-16 REPEATABLE READ 모드 요약

모드	REPEATABLE READ
LOCK	SELECT 문 – 공유락을 걸고 트랜잭션을 끝까지 유지함 UPDATE 문 – 배타락 설정함 다른 트랜잭션이 설정한 공유락은 읽지만 배타락은 읽지 못함
SQL 문	SET TRANSACTION ISOLATION LEVEL REPEATABLE READ
문제점	유령데이터 읽기

❖ SERIALIZABLE(Level = 3)

표 8-17 SERIALIZABLE 모드 요약

모드	SERIALIZABLE
LOCK	SELECT 문 – 공유락을 걸고 트랜잭션을 끝까지 유지함 UPDATE 문 – 배타락 설정함 다른 트랜잭션이 설정한 공유락은 읽지만 배타락은 읽지 못함 인덱스에 공유락을 설정하여 다른 트랜잭션의 INSERT 문이 금지됨
SQL 문	SET TRANSACTION ISOLATION LEVEL SERIALIZABLE
문제점	없음

3. MySQL의 트랜잭션

❖ 트랜잭션과 자동 커밋

- 다음과 같은 SQL 문을 실행하면 자동 커밋이 발생
(MySQL의 커밋은 자동 커밋이 기본값이다.

```
INSERT INTO users (id, name) VALUES (1, 'Park');
```

- 자동 커밋을 비활성화하려면 다음과 같이 설정함

```
SET autocommit = 0;
```

- 명시적으로 트랜잭션을 만드려면 트랜잭션의 시작과 끝을 나타내는 명령어 START TRANSACTION과 COMMIT 문을 사용한다.

```
START TRANSACTION;
-- 여러 쿼리 수행
COMMIT; -- 또는 ROLLBACK;
```

- DDL 명령어 CREATE, ALTER, DROP 명령어는 명령어 단위로 항상 커밋 된다.

3. MySQL의 트랜잭션

- 다음 예를 통하여 트랜잭션의 시작과 끝이 어디에 있는지 살펴보기
  MySQL은 자동 커밋을 기본값으로 활성화되어 시작함, SQL 문이 개별적인 트랜잭션으로 간주됨

```
DROP TABLE IF EXISTS Persons;
SET autocommit=1; -- autocommit=1 기본값에서 시작한다.
CREATE TABLE Persons(a INT, b CHAR (20));
-- create 문은 자동으로 commit 된다.
INSERT INTO Persons VALUES (10, 'Kim');
-- insert 문은 자동으로 commit 된다.

SET autocommit=0; -- 다음 여러 문장은 하나의 트랜잭션이 된다.
INSERT INTO Persons VALUES (15, 'Park');
INSERT INTO Persons VALUES (20, 'Lee');
DELETE FROM Persons WHERE b = 'Kim';
ROLLBACK; -- insert 2개와 delete 1개를 취소한다.
SELECT * FROM Persons;
-- 수행 결과는 (10, 'Kim')이 된다.
```

3. MySQL의 트랜잭션

❖ 의도 락

```
SELECT * FROM Book WHERE id = 1 FOR SHARE; -- 의도 공유락
SELECT * FROM Book WHERE id = 1 FOR UPDATE; -- 의도 배타락
```

```
-- 예약을 시도하는 사용자에게 항공기 123번 정보 제공
SELECT * FROM Flights WHERE airplane_id = 123;
-- 트랜잭션 시작
START TRANSACTION;
-- 해당 항공기의 남은 좌석 수 조회하고 의도 배타락으로 잠금 설정
SELECT remain_seats FROM Flights WHERE airplane_id = 123 FOR UPDATE;
-- 남은 좌석 수 체크
IF remaining_seats > 0 THEN
     -- 좌석을 예약하고 재고 감소
     INSERT INTO reservations(airplane_id, user_id) VALUES (123, 'user123');
     UPDATE Flights SET remain_seats = remain_seats - 1 WHERE airplane_id = 123;
     -- 예약 성공
     COMMIT;
ELSE -- 예약 실패 (남은 좌석이 없으므로 취소 작업 진행)
        ROLLBACK;
END IF;
```

3. MySQL의 트랜잭션

❖ MySQL READ UNCOMMITTED 고립 수준 실험

- 두 개의 트랜잭션 T1과 T2에 대하여 고립 수준을 완화하여 READ COMMITTED 모드로 실행할 경우 반복불가능 읽기 문제가 나타나는 상황을 마당서점 데이터베이스에서 실습해보기
- [표 8-18]에서 트랜잭션 T1은 Book 테이블에 저장된 도서 가격의 총액을 계산

표 8-18 반복불가능 읽기 문제 예(파일명: s06_t1.sql, s06_t2.sql)

트랜잭션 T1 READ COMMITTED 모드	트랜잭션 T2 READ COMMITTED 모드
SET TRANSACTION ISOLATION LEVEL READ COMMITTED; START TRANSACTION; USE madangdb; SELECT SUM(price) 총액 FROM Book; 총액 144500	

3. MySQL의 트랜잭션

	SET TRANSACTION ISOLATION LEVEL READ COMMITTED; START TRANSACTION; USE madangdb; SELECT SUM(price) 총액 FROM Book; 총액 144500 UPDATE Book SET price = price+500 WHERE bookid = 1; SELECT SUM(price) 총액 FROM Book; 총액 145000 COMMIT;
SELECT SUM(price) 총액 FROM Book; /* 앞의 결과와 다름 */ 총액 145000 COMMIT;	반복불가능 읽기 문제 발생

TIP 실습 시 첫 번째 문장 'SET TRANSACTION ISOLATION LEVEL READ COMMITTED;'을 생략할 경우 기본 고립 수준은 REPEATABLE READ다. 환경설정에서 고립수준을 바꿔줄 수 있다.

3. MySQL의 트랜잭션

- 반복불가능 읽기 문제를 방지하려면 트랜잭션 T1의 고립 수준을 REPEATABLE READ 모드로 상향하면 됨
- 고립 수준을 상향하면 T1은 락을 해지하지 않기 때문에 T2가 데이터를 UPDATE하는 것을 막을 수 있음
- T2가 UPDATE 문을 실행하려는 순간 대기 상태가 되고, TI의 트랜잭션이 커밋되기 전까지 최초 SNAPSHOT으로 SELECT를 수행하며 동일한 결과를 유지함

표 8-19 REPEATABLE READ 모드로 반복불가능 읽기 문제를 방지한 예(파일명: s07_t1.sql, s07_t2.sql)

트랜잭션 T1 REPEATABLE READ 모드	트랜잭션 T2 READ COMMITTED 모드
SET TRANSACTION ISOLATION LEVEL REPEATABLE READ; START TRANSACTION; USE madangdb; SELECT SUM(price) 총액 FROM Book; 총액 144500	

3. MySQL의 트랜잭션

	SET TRANSACTION ISOLATION LEVEL READ COMMITTED; START TRANSACTION; USE madangdb; SET SQL_SAFE_UPDATES=0; UPDATE Book SET price=price+500 WHERE bookid=1; SELECT SUM(price) 총액 FROM Book; 총액 145000 COMMIT;
SELECT SUM(price) 총액 FROM Book; /* 앞의 결과와 같음 */ 총액 144500 COMMIT;	

반복불가능 읽기 문제 없음

연습문제 (Q8.3)

14 다음 트랜잭션 T1, T2가 동시에 실행될 때 공유락과 배타락을 사용한다. 트랜잭션 T1에서 첫 번째 질의의 수행 결과는 20이고, 두 번째 질의의 수행 결과는 21이다. 아래 물음에 답하시오.

트랜잭션 T1	트랜잭션 T2
SELECT age FROM Users WHERE id=1;	
	UPDATE Users SET age=21 WHERE id=1; COMMIT;
SELECT age FROM Users WHERE id=1;	

(1) 이러한 현상이 일어나는 이유를 설명하고, 이 현상의 이름을 말하시오.

(2) (1)번의 현상을 방지하기 위한 방법을 설명하시오.

(3) (1)번의 현상을 방지하기 위한 SQL 문을 작성하시오.

04 회복

1. 회복의 개념
2. 회복과 트랜잭션
3. 로그 파일과 회복
4. 체크포인트와 회복

1. 회복의 개념

❖ 회복

- 데이터베이스에 장애가 발생했을 때 데이터베이스를 일관성 있는 상태로 되돌리는 DBMS의 기능

❖ 데이터베이스 시스템에서 발생할 수 있는 장애의 유형

- 시스템 충돌
- 미디어 장애
- 응용 소프트웨어 오류
- 자연재해
- 부주의 혹은 태업

- 데이터베이스 시스템에서 회복에 중점을 두는 유형은 시스템 충돌, 미디어 장애, 응용 소프트웨어 오류임
- 이러한 장애는 두 가지 결과를 초래함. 두 가지 장애로부터 데이터를 복구하는 방법을 살펴보기

2. 회복과 트랜잭션

- 트랜잭션은 데이터베이스 회복의 단위
- 장애가 발생하면 로그 내용을 참조하여 트랜잭션의 변경 내용을 모두 반영하거나 아예 반영하지 않는 방법으로 원자성을 보장함. 지속성도 마찬가지
- [그림 8-11]은 학습의 편의를 위하여 [그림 8-1](449쪽)을 다시 인용한 것

[트랜잭션 수행 과정] ① - ② - ③ - ④ - COMMIT(부분 완료) - ⑤ - ⑥ - COMMIT(완료)

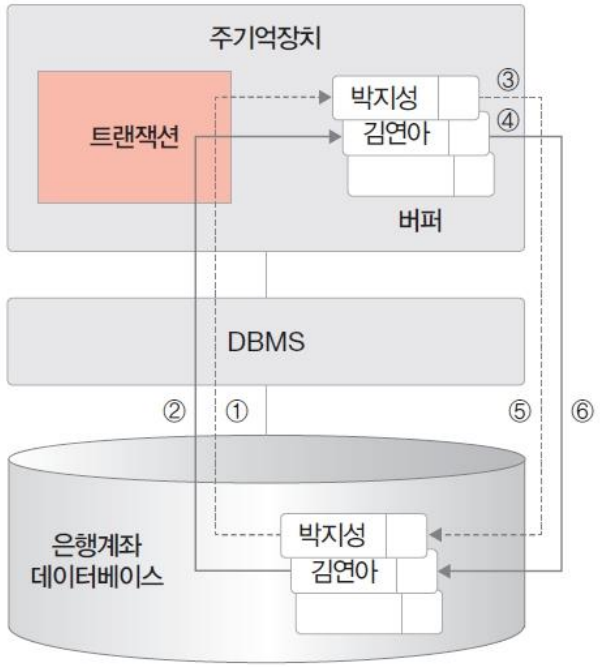
```
START TRANSACTION
① /* 박지성 계좌를 읽어 온다 */
② /* 김연아 계좌를 읽어 온다 */
   /* 잔고 확인 */

③ /* 예금인출 박지성 */
UPDATE   Customer
SET      balance=balance-10000
WHERE    name='박지성';

④ /* 예금입금 김연아 */
UPDATE   Customer
SET      balance=balance+10000
WHERE    name='김연아';

COMMIT /* 부분 완료 */
⑤ /* 박지성 계좌를 기록한다 */
⑥ /* 김연아 계좌를 기록한다 */
```

(a) 계좌이체 트랜잭션



(b) 트랜잭션 수행 과정

그림 8-11 계좌이체 트랜잭션과 수행 과정

(DBMS가 즉시 갱신하는 경우 COMMIT전에 ⑤⑥이 DB에 기록될 수도 있음)

2. 회복과 트랜잭션

- 트랜잭션의 수행 과정을 상태도와 함께 나타내면 [그림 8-12]와 같음

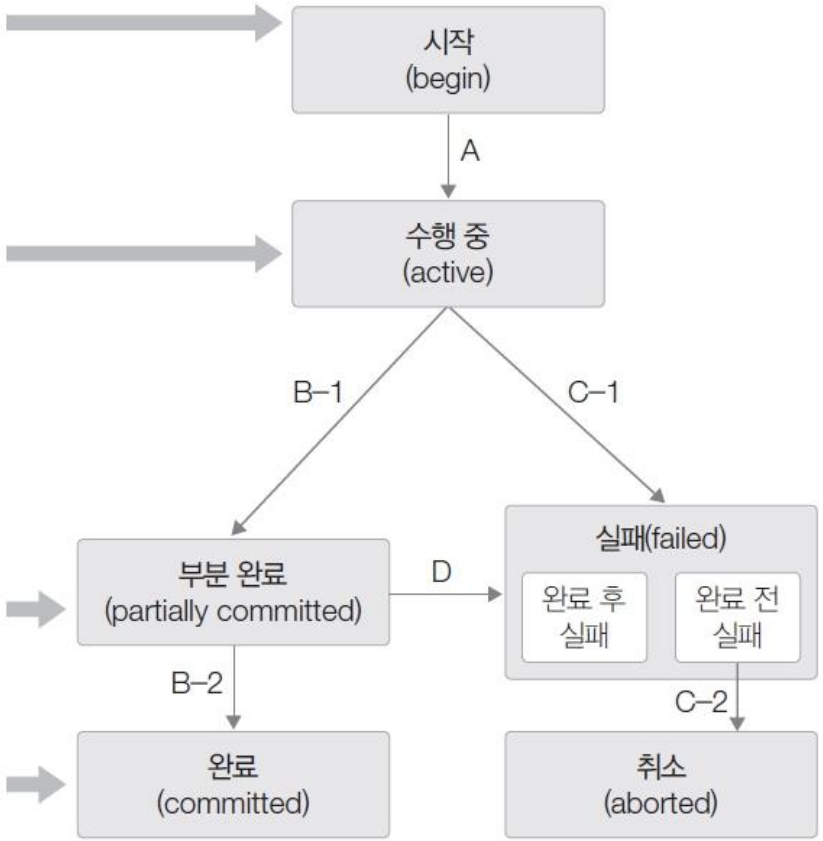
```
START TRANSACTION
① /* 박지성 계좌를 읽어 온다 */
② /* 김연아 계좌를 읽어 온다 */
   /* 잔고 확인 */

③ /* 예금인출 박지성 */
UPDATE   Customer
SET      balance=balance-10000
WHERE    name='박지성';

④ /* 예금입금 김연아 */
UPDATE   Customer
SET      balance=balance+10000
WHERE    name='김연아';

COMMIT /* 부분 완료 */
⑤ /* 박지성 계좌를 기록한다 */
⑥ /* 김연아 계좌를 기록한다 */
```

(a) 계좌이체 트랜잭션



(b) 트랜잭션 상태도

그림 8-12 트랜잭션 수행과 상태도

3. 로그 파일과 회복

❖ 로그 파일의 개념

- DBMS는 트랜잭션이 수행 중이거나 수행 종료 후 발생하는 데이터베이스 손실을 방지하기 위해 로그 파일을 사용
- 로그 파일에 저장된 로그의 구조

  <트랜잭션 번호, 로그 타입, 데이터 항목 이름, 수정 전 값, 수정 후 값>

- 예) 계좌이체 프로그램의 경우 로그 파일에 저장되는 내용

```
<T1, START>
<T1, UPDATE, Customer (박지성).balance, 100000, 90000>
<T1, UPDATE, Customer (김연아).balance, 100000, 110000>
<T1, COMMIT>
```

3. 로그 파일과 회복

- [그림 8-13]의 (b)는 로그 파일을 포함한 트랜잭션의 수행 과정을 나타냄

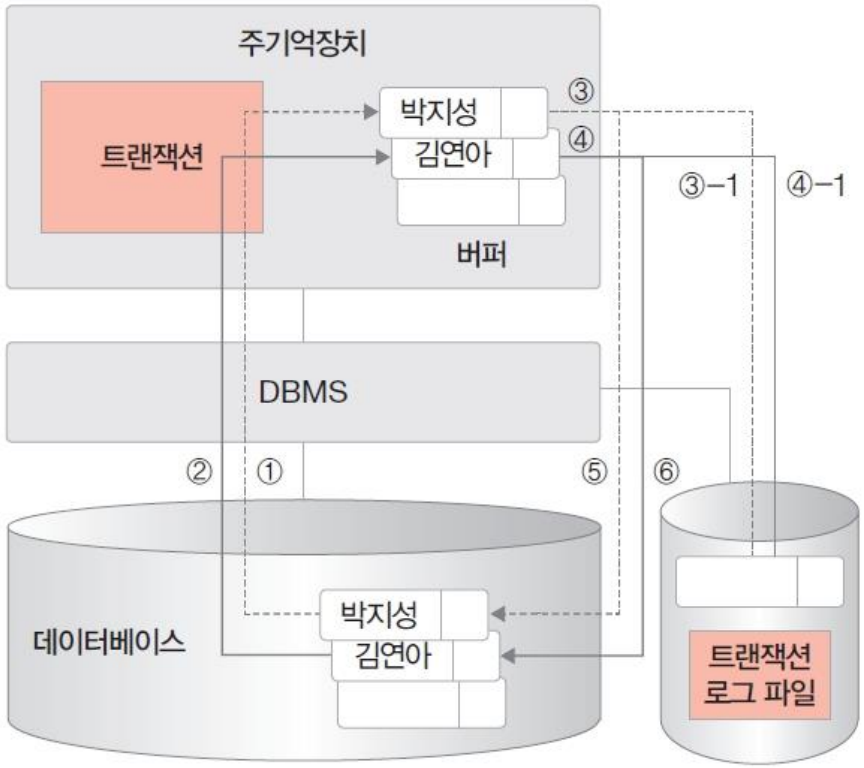
```
START TRANSACTION
① /* 박지성 계좌를 읽어 온다 */
② /* 김연아 계좌를 읽어 온다 */
   /* 잔고 확인 */

③ /* 예금인출 박지성 */
UPDATE   Customer
SET      balance=balance-10000
WHERE    name='박지성';

④ /* 예금입금 김연아 */
UPDATE   Customer
SET      balance=balance+10000
WHERE    name='김연아';

COMMIT /* 부분 완료 */
⑤ /* 박지성 계좌를 기록한다 */
⑥ /* 김연아 계좌를 기록한다 */
```

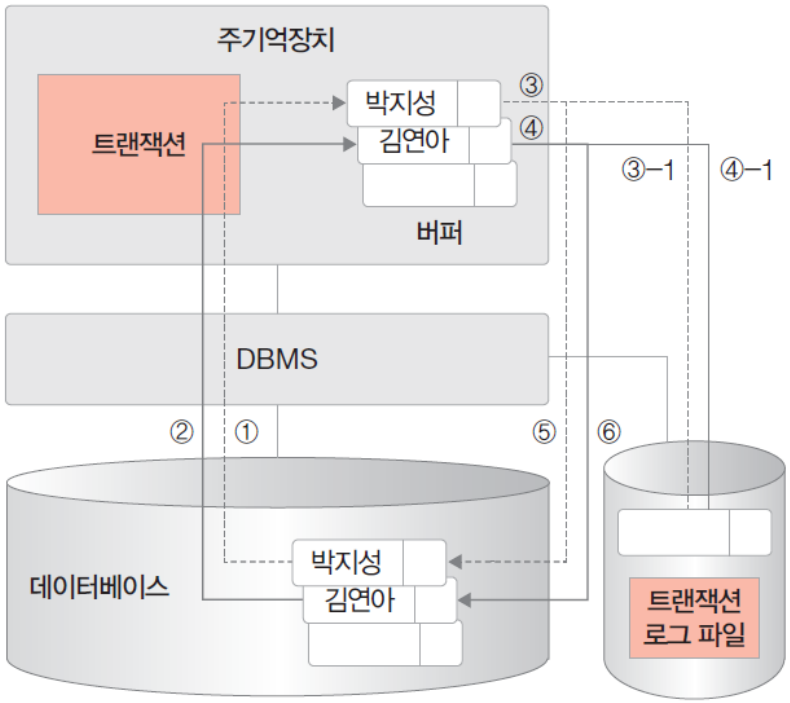
(a) 계좌이체 트랜잭션



(b) 트랜잭션 수행 과정

그림 8-13 트랜잭션 수행과 로그 파일

3. 로그 파일과 회복



(b) 트랜잭션 수행 과정

❖ 트랜잭션 수행 가능 시나리오(③-2, ④-2 추가하여 설명)

③-2 : 박지성 계좌 버퍼의 내용을 DB에 쓰기

④-2 : 김연아 계좌 버퍼의 내용을 DB에 쓰기

❖ 즉시갱신 : COMMIT 전에도 DB에 변경사항 반영 가능

(즉시갱신 시나리오 1)

①, ②, ③ , ③-1, ③-2, ④, ④-1, ④-2, COMMIT

(즉시갱신 시나리오 2)

①, ②, ③ , ④, ④-1, ③-1, ④-2, ③-2, COMMIT

(즉시갱신 시나리오 3)

①, ②, ③ , ④, ④-1, ③-1, ③-2, COMMIT, ④-2

(즉시갱신 시나리오 4)

①, ②, ③ , ④, ④-1, ③-1, COMMIT, ③-2, ④-2

❖ 지연갱신 : COMMIT 후에만 DB에 변경사항 반영 가능

(지연갱신 시나리오 1)

①, ②, ③ , ④, ④-1, ③-1, ③-2, COMMIT, ③-2, ④-2

3. 로그 파일과 회복

❖ 로그 파일을 이용한 회복

- 데이터의 변경 기록을 저장해 둔 로그 파일을 이용하면 시스템 장애도 복구할 수 있음

- 예) 다음과 같이 두 개의 트랜잭션이 실행
  - 아래 두 개의 트랜잭션이 실행된다고 하자. 편의상 트랜잭션의 연산 SELECT, UPDATE는 read_item( ), write_item( )으로 대체한다. 트랜잭션은 각각 데이터 A, B, C, D를 읽거나 쓰는 작업을 진행한다. 데이터(A, B, C, D)의 초깃값은 (100, 200, 300, 400)이다.

트랜잭션 T1	트랜잭션 T2

read_item(A);
A = A+10;
read_item(B);
B = B+10;
write_item(B);
read_item(C);
C = C+10;
write_item(C);
write_item(A);
	read_item(A);
	A = A+10;
	write_item(A);
	read_item(D);
	D = D+10;
	read_item(B)
	B = B+10;
	write_item(B);
	write_item(D);

3. 로그 파일과 회복

- 트랜잭션이 T1 → T2 순으로 실행된다면 [그림 8-14]와 같은 로그 파일이 생성됨

로그 번호	로그 레코드
1	[T1, START]
2	[T1, UPDATE, B, 200, 210]
3	[T1, UPDATE, C, 300, 310]
4	[T1, UPDATE, A, 100, 110]
5	[T1, COMMIT]
6	[T2, START]
7	[T2, UPDATE, A, 110, 120]
8	[T2, UPDATE, B, 210, 220]
9	[T2, UPDATE, D, 400, 410]
10	[T2, COMMIT]

트랜잭션 T1	트랜잭션 T2
read_item(A);	read_item(A);
A = A+10;	A = A+10;
read_item(B);	write_item(A);
B = B+10;	read_item(D);
write_item(B);	D = D+10;
read_item(C);	read_item(B)
C = C+10;	B = B+10;
write_item(C);	write_item(B);
write_item(A);	write_item(D);

그림 8-14 트랜잭션 T1, T2의 로그 파일 예

3. 로그 파일과 회복

- **DBMS는 트랜잭션이 종료되었는지 혹은 중단되었는지 여부를 판단하여 종료된 트랜잭션은 종료를 확정하기 위하여 재실행(REDO)을 진행하고, 중단된 트랜잭션은 없던 일로 되돌리기 위해 취소(UNDO)를 진행한다.**
  (트랜잭션의 원자성 all(REDO) or nothing(UNDO)을 수행한다)
- **트랜잭션의 재실행(REDO)**
  - 장애가 발생한 후 시스템을 다시 가동했을 때, 로그 파일에 트랜잭션의 시작 START과 함께 종료 COMMIT가 있는 경우임
  - COMMIT 연산이 로그에 있다는 것은 트랜잭션이 모두 완료되었다는 의미
  - 로그를 보면서 트랜잭션이 변경한 내용을 데이터베이스에 다시 기록하는 과정이 필요
- **트랜잭션의 취소(UNDO)**
  - 장애가 발생한 후 시스템을 다시 가동했을 때, 로그 파일에 트랜잭션의 시작 START만 있고 종료 COMMIT가 없는 경우임
  - COMMIT 연산이 로그에 보이지 않는다는 것은 트랜잭션이 완료되지 못했다는 의미로, 트랜잭션이 한 일을 모두 취소해야 함
  - 이 경우 완료하지 못했지만, 버퍼의 변경 내용이 데이터베이스에 기록되어 있을 가능성이 있으므로 로그를 보면서 트랜잭션이 변경한 내용을 데이터베이스에서 원상 복구시켜야 함

3. 로그 파일과 회복

- [그림 8-14]의 로그 번호 i를 따라 회복 작업 과정을 살펴보면 [표 8-20]과 같음

표 8-20 트랜잭션 로그와 회복 방법(즉시갱신 방법)

로그 번호	작업	결과
i=0	아무 작업도 할 필요가 없음	T1과 T2가 수행을 시작하지 않았음
1<= i<=4	UNDO(T1): T1 취소 → T1이 i까지 생성한 로그 레코드를 이용하여 데이터베이스 항목을 되돌림	T1을 수행하지 않은 것과 같음
5<= i<=9	REDO(T1): T1 재수행 → 1부터 4까지 T1이 생성한 로그 레코드를 이용하여 데이터베이스 항목 값을 기록함 UNDO(T2): T2 취소 → T2가 5부터 i까지 생성한 로그 레코드를 이용하여 데이터베이스 항목을 되돌림	T1은 수행이 완료됨 T2는 수행하지 않은 것과 같음
10	REDO(T1): T1 재수행 REDO(T2): T2 재수행	T1, T2는 수행이 완료됨

3. 로그 파일과 회복

- 지금까지 설명한 방법은 부분 완료 전에라도 트랜잭션이 변경한 내용 일부가 데이터베이스에 기록될 수 있음을 가정한 것. 이를 즉시갱신이라고 함
- 트랜잭션이 반드시 부분 완료된 후 변경 내용을 데이터베이스에 기록하는 방법도 있음. 이를 지연갱신이라고 함

❖ 즉시갱신(immediate update) (REDO-UNDO)

- '버퍼 → 로그 파일', '버퍼 → 데이터베이스' 작업이 부분 완료 전에 동시에 진행될 수 있음

❖ 지연갱신(deferred update) (REDO, NO-UNDO)

- '버퍼 → 로그 파일'이 모두 끝난 후에 부분 완료를 하고 이후 '버퍼→ 데이터베이스' 작업을 진행하는 방법

3. 로그 파일과 회복

- [그림 8-14]의 예에 지연갱신 방법을 사용하면 [표 8-21]과 같음

표 8-21 트랜잭션 로그와 회복 방법(지연갱신 방법)

로그 번호	작업	결과
i=0	아무 작업도 할 필요가 없음	T1과 T2가 수행을 시작하지 않았음
1<=i<=4	T1: 아무 작업도 할 필요가 없음	T1을 수행하지 않은 것과 같음
5<=i<=9	REDO(T1): T1 재수행 → 1부터 4까지 T1이 생성한 로그 레코드를 이용하여 데이터 베이스 항목 값을 기록함 T2: 아무 작업도 할 필요가 없음	T1은 수행이 완료됨 T2는 수행하지 않은 것과 같음
10	REDO(T1): T1 재수행 REDO(T2): T2 재수행	T1, T2는 수행이 완료됨

4. 체크포인트와 회복

❖ 체크포인트

- 회복 시 많은 양의 로그를 검색하고 갱신하는 시간을 줄이기 위하여 몇십 분 단위로 데이터베이스와 트랜잭션 로그 파일을 동기화한 후 동기화한 시점을 로그 파일에 기록해 두는 방법 혹은 그 시점

❖ 체크포인트 시점에는 다음과 같은 작업을 진행

- 주기억장치의 로그 레코드를 모두 하드디스크의 로그 파일에 저장
- 버퍼에 있는 변경된 내용을 하드디스크의 데이터베이스에 저장(즉시갱신의 경우)
- 체크포인트를 로그 파일에 표시

❖ 트랜잭션의 로그 기록에 따라 회복하는 방법은 다음과 같음

- **체크포인트 이전에 COMMIT 기록이 있는 트랜잭션의 경우**
  아무 작업이 필요 없음. 로그에 체크포인트가 나타나는 시점은 이미 변경 내용이 데이터베이스에 모두 기록된 후이기 때문.
- **체크포인트 이후에 COMMIT 기록이 있는 트랜잭션의 경우**
  REDO(T)를 진행.
- **체크포인트 이후에 COMMIT 기록이 없는 트랜잭션의 경우**
  즉시 갱신 방법을 사용했다면 UNDO(T)를 진행.
  지연 갱신 방법을 사용했다면 아무것도 할 필요가 없음.

4. 체크포인트와 회복

- 그림 8-15 상황에서 시스템 장애 시
- **즉시갱신 방법**
  - T2, T3에는 어떤 작업도 할 필요가 없음
  - T4, T5에는 REDO를 진행하고
  - T1, T6에는 UNDO를 진행해야 함
- **지연갱신 방법**
  - T2, T3에는 어떤 작업도 할 필요가 없음
  - T4, T5에는 REDO를 진행해야 함
  - T1, T6에는 어떤 작업도 필요 없음

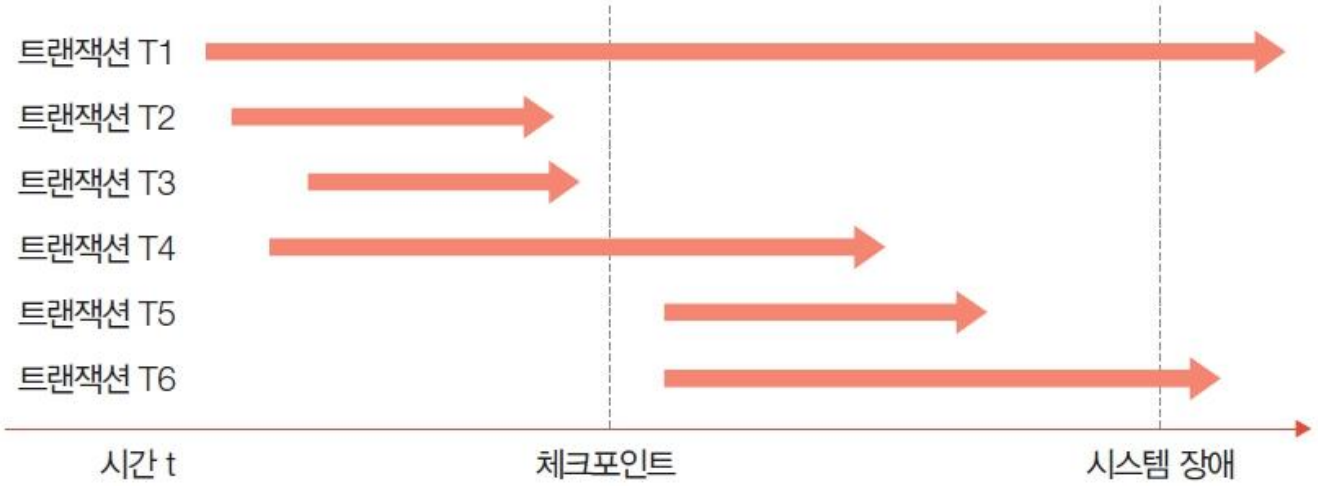


그림 8-15 트랜잭션 로그 기록과 체크포인트

4. 체크포인트와 회복

- 로그 기록을 사용한 예제

- 트랜잭션 T1, T2, T3가 동시에 실행된 후 [그림 8-16]과 같이 로그 기록을 남김
- 즉시 갱신 기법을 사용하여 회복을 한다면 REDO(T2), UNDO(T3)가 진행된다.
  T1에 대해서는 아무 작업이 필요 없다.

로그 번호	로그 레코드
1	[T1, START]
2	[T1, UPDATE, B, 200, 120]
3	[T1, UPDATE, C, 300, 310]
4	[T2, START]
5	[T2, UPDATE, A, 110, 120]
6	[T1, UPDATE, A, 120, 110]
7	[T1, COMMIT]
8	[T2, UPDATE, B, 120, 220]
9	[CHECKPOINT]
10	[T3, START]
11	[T3, UPDATE, A, 110, 120]
12	[T2, UPDATE, D, 400, 410]
13	[T2, COMMIT]
14	[T3, UPDATE, B, 220, 230]
15	~~ 시스템 장애 ~~

그림 8-16 체크포인트가 포함된 로그 기록

연습문제 (Q8.4)

10 회복을 위한 로그 기록 방법 중 지연갱신에서 사용하는 방법은?

① UNDO/REDO

② UNDO/NO-REDO

③ NO-UNDO/REDO

④ NO-UNDO/NO-REDO

18 오른쪽 표는 어느 트랜잭션 T1, T2, T3의 로그 기록이다. 지연갱신 방법을 사용한다고 가정한다. 15번 로그에서 시스템 장애가 일어났을 때 각 트랜잭션 T1, T2, T3를 복구하는 방법에 대하여 설명하시오.

로그 번호	로그 레코드
1	[T1, START]
2	[T1, UPDATE, B, 200, 120]
3	[T1, UPDATE, C, 300, 310]
4	[T2, START]
5	[T2, UPDATE, D, 110, 120]
6	[T1, UPDATE, A, 120, 110]
7	[T1, COMMIT]
8	[T2, UPDATE, E, 120, 220]
9	[CHECKPOINT]
10	[T3, START]
11	[T3, UPDATE, G, 110, 120]
12	[T2, UPDATE, F, 400, 410]
13	[T2, COMMIT]
14	[T3, UPDATE, H, 220, 230]
15	~~ 시스템 장애 ~~

요약

1. 트랜잭션의 상태도
2. 트랜잭션의 성질
3. 동시성 제어
4. 갱신 손실
5. 락
6. 2단계 락킹
7. 데드락
8. 트랜잭션 동시 실행 문제
9. 트랜잭션 고립 수준 명령어
10. 로그 파일을 이용한 회복
11. 회복을 위한 로그 기록 방법
12. 체크포인트