원리를 알면 IT가 맛있다
IT COOKBOOK

IT CookBook, SQL Server로 배우는 데이터베이스 개론과 실습

[강의교안 이용 안내]

Chapter8.
트랜잭션, 동시성 제어, 회복

SQL Server로 배우는 데이터베이스 개론과 실습

한빛아카데미
Hanbit Academy, Inc.

목차

학습목표

- 트랜잭션의 개념을 이해하고 데이터베이스에서 왜 필요한지 알아본다.
- 트랜잭션 실행 시 동시성 제어가 필요한 이유를 알아보고 락킹을 이용한 동시성 제어 기법에 대해 알아본다.
- 락킹보다 완화된 방법으로 트랜잭션의 동시성을 높이는 트랜잭션 고립 수준에 대해 알아본다.
- 데이터베이스 시스템에 문제가 생길 경우 복구하는 방법을 알아본다.

01. 트랜잭션

- 트랜잭션의 개념
- 트랜잭션의 성질
- 트랜잭션과 DBMS

1.1 트랜잭션

- **트랜잭션(transaction)은 DBMS에서 데이터를 다루는 논리적인 작업의 단위다.**

- **데이터베이스에서 트랜잭션을 정의하는 이유**
  - 데이터베이스에서 데이터를 다룰 때 장애가 일어나는 경우가 있다. 트랜잭션은 장애 시 데이터를 복구하는 작업의 단위가 된다.
  - 데이터베이스에서 여러 작업이 동시에 같은 데이터를 다룰 때가 있다. 트랜잭션은 이 작업을 서로 분리하는 단위가 된다.

- **트랜잭션은 전체가 수행되거나 또는 전혀 수행되지 않아야 한다(all or nothing).**

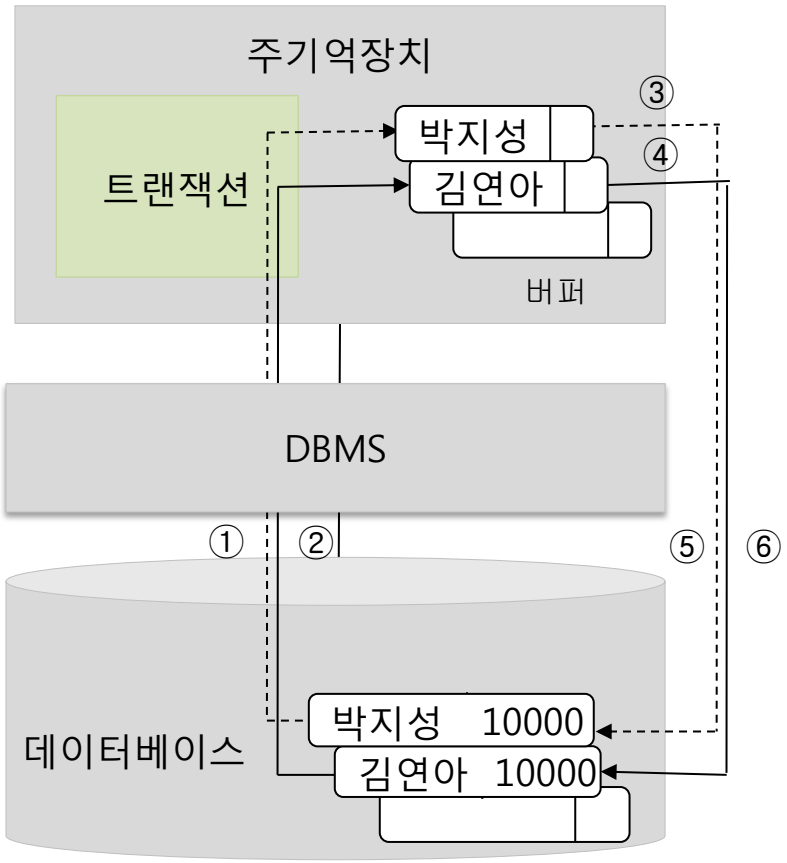
EX)은행 업무를 보는데 A 계좌(박지성)에서 B 계좌(김연아)로 10,000원을 이체할 경우

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

1.1 트랜잭션


```
BEGIN TRANSACTION
① /* 박지성 계좌를 읽어온다 */
② /* 김연아 계좌를 읽어온다 */
   /* 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 /* 부분완료 */
⑤ /* 박지성 계좌를 기록한다 */
⑥ /* 김연아 계좌를 기록한다 */

COMMIT TRANSACTION
```

(a) 계좌이체 트랜잭션



(b) 트랜잭션 수행과정

그림 8-1 **계좌이체 트랜잭션과 수행 과정**

1.1 트랜잭션

트랜젝션 수행 과정

① A 계좌(박지성)의 값을 하드디스크(데이터베이스)에서 주기억장치 버퍼로 읽어온다
② B 계좌(김연아)의 값을 하드디스크(데이터베이스)에서 주기억장치 버퍼로 읽어온다.
③ A 계좌(박지성)에서 10,000원을 인출한 값을 저장한다.
④ B 계좌(김연아)에 10,000원을 입금한 값을 저장한다.
⑤ A 계좌(박지성)의 값을 주기억장치 버퍼에서 하드디스크(데이터베이스)에 기록한다.
⑥ B 계좌(김연아)의 값을 주기억장치 버퍼에서 하드디스크(데이터베이스)에 기록한다.

트랜잭션의 종료(COMMIT)를 알리는 방법

- [방법 1] ①-②-③-④-COMMIT-⑤-⑥
- [방법 2] ①-②-③-④-⑤-⑥-COMMIT

→ DBMS는 사용자에게 빠른 응답성을 보장하기 위해 [방법 1]을 선택한다.

그림 8-2 **트랜잭션의 수행 과정**

1.2 트랜잭션의 성질

표 8-1 트랜잭션과 프로그램의 차이점

구분	트랜잭션	프로그램
프로그램 구조	BEGIN TRANSACTION ... COMMIT TRANSACTION	main() { ... }
다루는 데이터	데이터베이스 저장된 데이터	파일에 저장된 데이터
번역기	DBMS	컴파일러
성질	원자성, 일관성, 고립성, 지속성	-

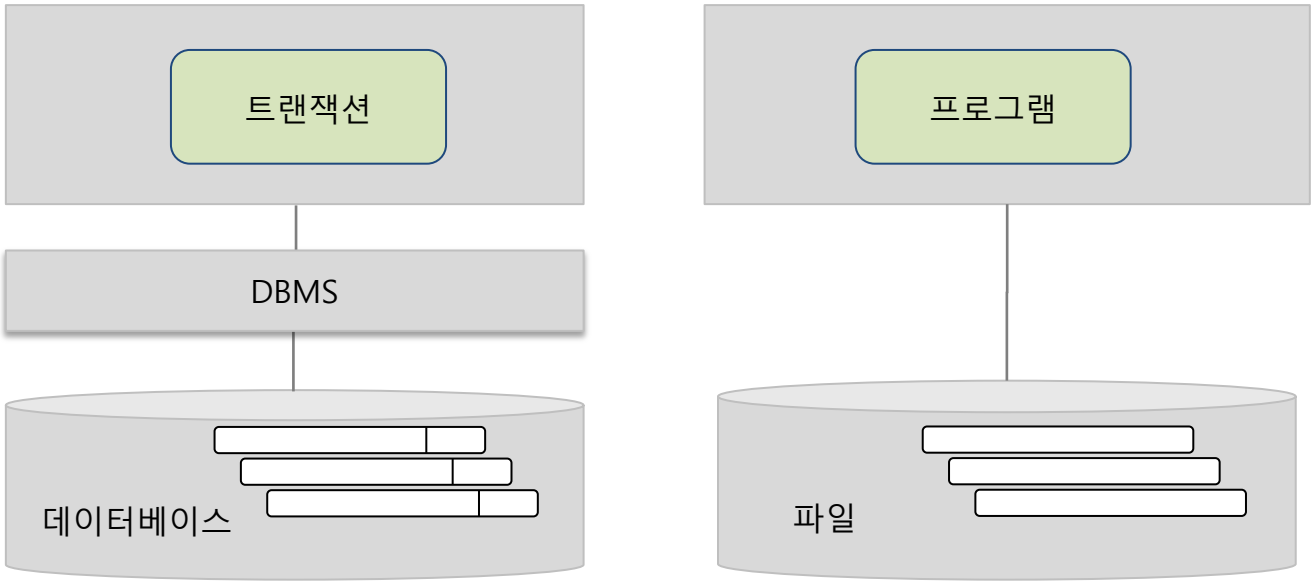


그림 8-3 컴퓨터 시스템 내의 트랜잭션과 프로그램

1.2 트랜잭션의 성질

트랜잭션의 ACID 성질

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

1.2.1 원자성

- **원자성(Atomicity)이란 트랜잭션이 원자처럼 더 이상 쪼개지지 않는 하나의 프로그램 단위로 동작해야 한다는 의미다. 즉 일부만 수행되는 일이 없도록 전부 수행하거나 아예 수행하지 않아야(all or nothing) 하는 성질이다.**

표 8-3 트랜잭션 제거 명령어(TCL)

명령어	문법	설명
BEGIN	BEGIN { TRAN | TRANSACTION }	트랜잭션의 시작을 표시
COMMIT	COMMIT { TRAN | TRANSACTION }	트랜잭션의 종료를 표시
ROLLBACK	ROLLBACK { TRAN | TRANSACTION } [<savepoint>]	트랜잭션을 전체 혹은 <savepoint>까지 무효화시킴
SAVE	SAVE { TRAN | TRANSACTION } { <savepoint> }	<savepoint>를 만듬

1.2.1 원자성

- **다음은 T-SQL에서 트랜잭션을 선언하고 수행하는 예다.**

```
BEGIN TRANSACTION
        INSERT INTO Book(bookid, bookname) VALUES (100, 'Database');
        SAVEPOINT a;
        INSERT INTO Customer(custid, name) VALUES (100, '홍길동');
        SAVEPOINT b;
        INSERT INTO Orders(orderid, custid, bookid) VALUES (100, 100, 100);
        ...
        /* a로 롤백하면 Customer와 Orders 테이블에 삽입은 없던 일이 된다 */
        IF (...) ROLLBACK to a;
        ...
        /* b로 롤백하면 b부터 현재까지 작업은 없던 일이 된다 */
        IF (...) ROLLBACK to b;
        ...
        /* ROLLBACK 명령문을 수행하면 트랜잭션 작업 전체가 없던 일이 된다 */
        IF (...) ROLLBACK;
        ...
/* 이제까지 진행한 작업을 데이터베이스에 반영한다 */
COMMIT;
```

1.2.2 일관성

- **트랜잭션은 데이터베이스의 일관성Consistency을 유지해야 한다. 일관성은 테이블이 생성될 때 CREATE 문과 ALTER 문의 무결성 제약조건을 통해 명시된다.**

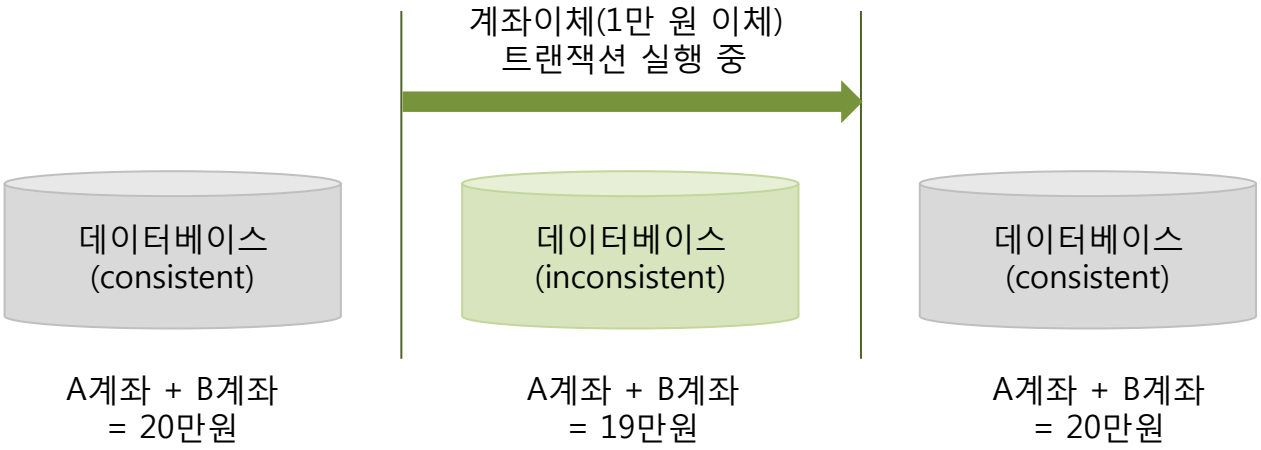


그림 8-4 데이터베이스 변경 중과 변경 후의 일관성

1.2.3 고립성

- 데이터베이스는 공유가 목적이기 때문에 여러 트랜잭션이 동시에 수행된다. 동시에 수행되는 트랜잭션은 상호 존재를 모르고 독립적으로 수행되는데, 이를 고립성 Isolation이라고 한다. 고립성을 유지하기 위해서는 트랜잭션이 변경 중인 임시 데이터를 다른 트랜잭션이 읽고 쓸 때 제어가 필요하다.

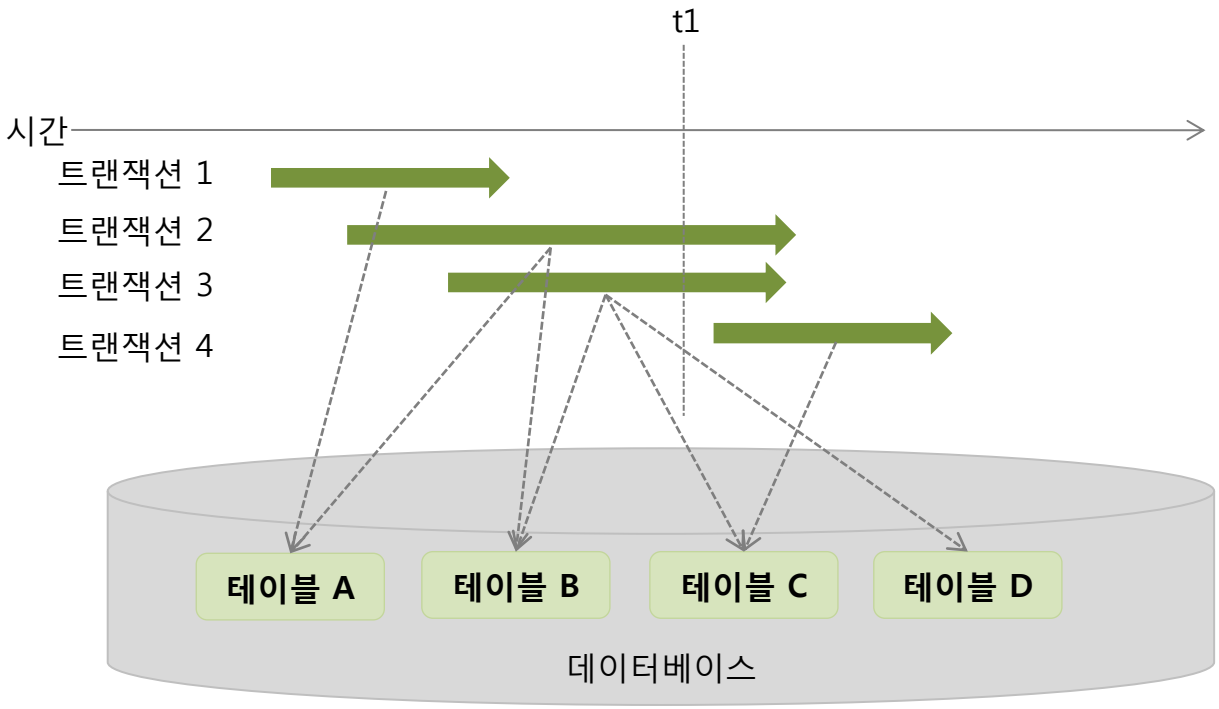


그림 8-5 트랜잭션의 동시 수행과 데이터 공유

1.2.4 지속성

- **트랜잭션이 정상적으로 완료(commit) 혹은 부분완료(partial commit)한 데이터는 DBMS가 책임지고 데이터베이스에 기록한다. 이러한 성질을 트랜잭션의 지속성(Durability)이라고 한다. DBMS 복구 시스템은 트랜잭션이 작업한 내용을 수시로 로그(log) 데이터베이스에 기록하였다가 문제가 발생하면 로그 파일을 이용하여 복구 작업을 수행한다.**

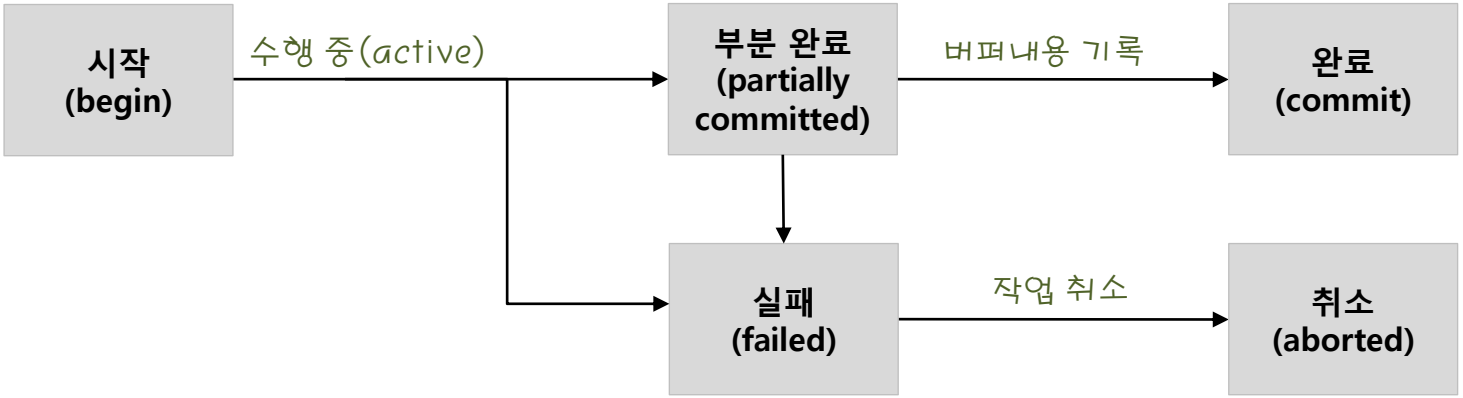


그림 8-6 **트랜잭션의 상태도**

- 부분완료(partially committed) : 트랜잭션 수행은 완료되었지만 변경 내용이 데이터베이스에 기록되었는지 확실하지 않은 상태다. 이 상태에서는 DBMS가 최종적으로 변경 내용을 데이터베이스에 기록해야 완료(committed) 상태가 된다. 만약 시스템 내부 문제 혹은 시스템 다운 등으로 DBMS가 변경 내용을 데이터베이스에 기록하지 못하면 실패(failed) 상태가 된다.
- 실패(failed) : 트랜잭션을 중간에 중단하였거나, 부분완료 상태에서 변경 내용을 데이터베이스에 저장하지 못한 상태를 말한다. 실패 상태에서 DBMS는 트랜잭션이 수행한 작업을 모두 원상복구시킨다.

1.3 트랜잭션과 DBMS

- DBMS는 원자성을 유지하기 위해 회복(복구) 관리자 프로그램을 작동시킨다.
- DBMS는 일관성을 유지하기 위해 무결성 제약조건을 활용한다.
- DBMS는 고립성을 유지하기 위해 일관성을 유지하는 것과 마찬가지로 동시성 제어 알고리즘을 작동시킨다.
- DBMS는 지속성을 유지하기 위해 회복 관리자 프로그램을 이용한다.

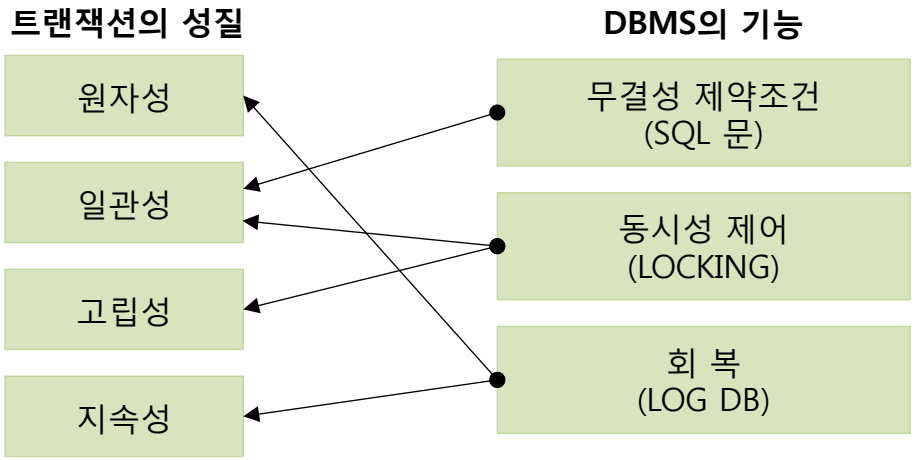


그림 8-7 트랜잭션의 성질과 DBMS의 기능

02. 동시성 제어

- 갱신손실 문제
- 락

02 동시성 제어

- **트랜잭션이 동시에 수행될 때, 일관성을 해치지 않도록 트랜잭션의 데이터 접근을 제어하는 DBMS의 기능을 동시성 제어(concurrency control)라고 한다.**

표 8-3 트랜잭션의 읽기(read)/쓰기(write) 시나리오

	트랜잭션1	트랜잭션2	발생 문제	처리 방법
[상황 1]	읽기	쓰기	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)

2.1 갱신손실 문제

- **갱신손실(lost update) 문제는 두 개의 트랜잭션이 한 개의 데이터를 동시에 갱신(update)할 때 발생한다. 갱신손실 문제는 데이터베이스에서 절대 발생하면 안 되는 현상이다.**

- **[작업 설명] 한 개의 데이터에 두 개의 트랜잭션이 접근하여 갱신하는 작업**

- **[시나리오] 두 개의 트랜잭션이 동시에 작업을 진행**

- **[문제 발생] 갱신손실**

  T2는 잘못된 데이터로 작업하여 잘못된 결과를 만든 다음 T1의 갱신 작업을 무효화하고 덧쓰기를 수행한 것이다. T1의 갱신이 손실된 갱신손실(lost update) 문제가 발생한 것이다.

2.1 갱신손실 문제

트랜잭션 TI	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

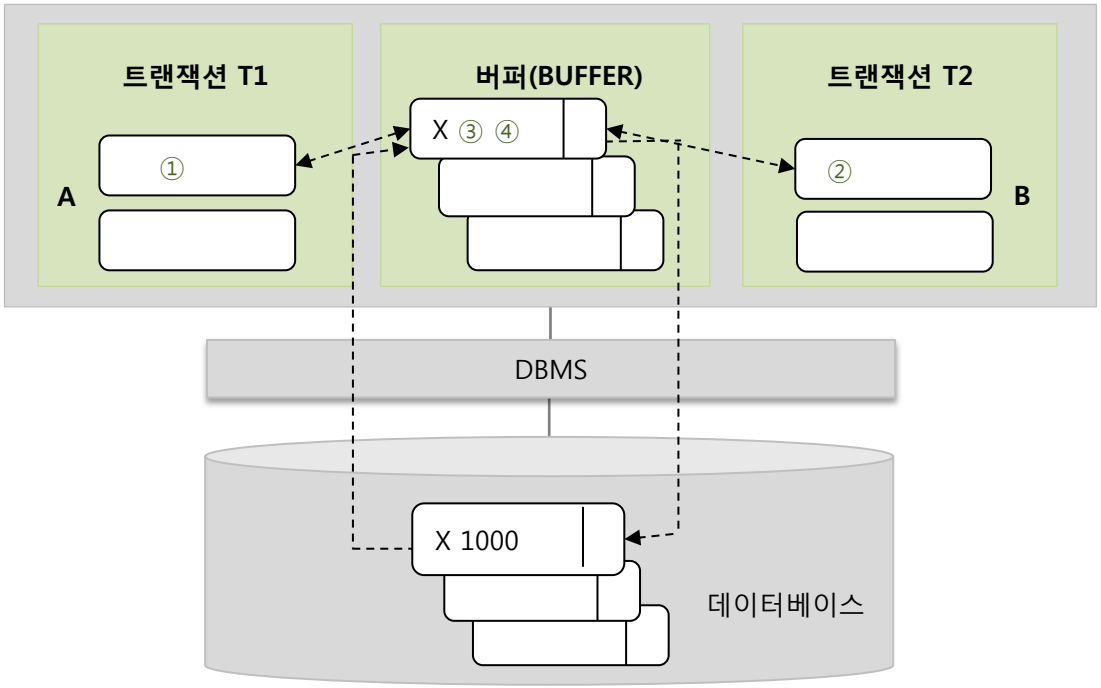


그림 8-8 갱신손실 문제 발생 시나리오

2.2 락

- 갱신손실 문제를 해결하려면 상대방 트랜잭션이 데이터를 사용하는지 여부를 알 수 있는 규칙이 필요하다. 즉 자신이 데이터를 수정 중이라는 사실을 알리면 된다. 알리는 방법으로 락(lock)이라는 잠금장치를 사용한다.

2.2.1 락의 개념

TI	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

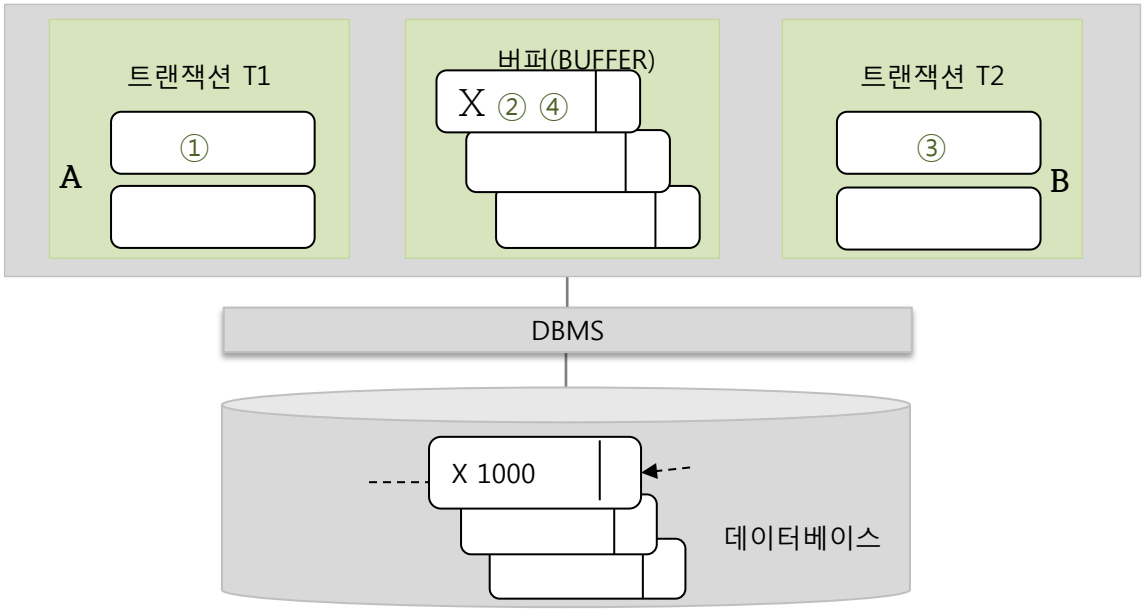


그림 8-9 락을 이용한 갱신손실 문제 해결

2.2.1 락의 개념

- **[작업 설명] 한 개의 데이터에 두 개의 트랜잭션이 접근하여 갱신하는 작업**

트랜잭션 T1	트랜잭션 T2
BEGIN TRAN; SELECT * FROM Book WHERE bookid=1; UPDATE Book SET price=7100 WHERE bookid=1; SELECT * FROM Book WHERE bookid=1; COMMIT;	BEGIN TRAN; SELECT * FROM Book WHERE bookid=1; UPDATE Book SET price=price+100 WHERE bookid=1; SELECT * FROM Book WHERE bookid=1; COMMIT;

2.2.1 락의 개념

- [시나리오] SQL Server에서 두 개의 갱신 트랜잭션을 동시에 실행

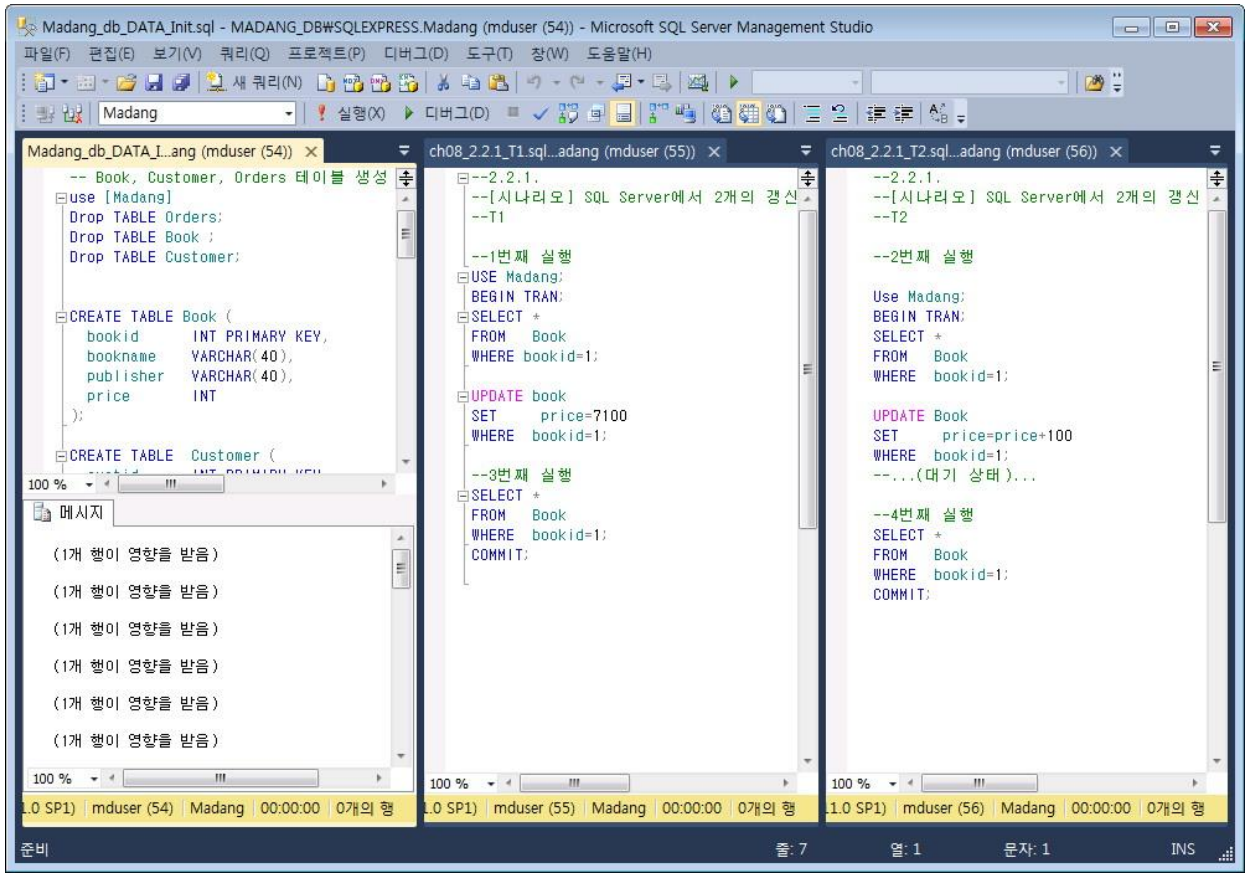


그림 8-10 SQL Server에서 두 개의 트랜잭션을동시에 실행시키는 화면

트랜잭션 T1	트랜잭션 T2
BEGIN TRAN; USE Madang; SELECT * FROM Book WHERE bookid=1; UPDATE Book SET price=7100 WHERE bookid=1; bookid | bookname | publisher | price 1 | 축구의 역사 | 굿스포츠 | 7100	
	BEGIN TRAN; Use Madang; SELECT * FROM Book WHERE bookid=1; UPDATE Book SET price=price+100 WHERE bookid=1; ...(대기 상태)...
SELECT * FROM Book WHERE bookid=1; COMMIT; bookid | bookname | publisher | price 1 | 축구의 역사 | 굿스포츠 | 7100	/ * COMMIT과 동시에 결과가 나타남 */ bookid | bookname | publisher | price 1 | 축구의 역사 | 굿스포츠 | 7100
	SELECT * FROM Book WHERE bookid=1; COMMIT; bookid | bookname | publisher | price 1 | 축구의 역사 | 굿스포츠 | 7200

2.2.2 락의 유형

- **락은 트랜잭션이 읽기를 할 때 사용하는 락인 공유락(LS, shared lock)과 읽고 쓰기를 할 때 사용하는 배타락(LX, exclusive lock)으로 나뉜다.**

- **공유락과 배타락을 사용하는 규칙**
  - 데이터에 락이 걸려있지 않으면 트랜잭션은 데이터에 락을 걸 수 있다.
  - 트랜잭션이 데이터 X를 읽기만 할 경우 LS(X)를 요청하고, 읽거나 쓰기를 할 경우 LX(X)를 요청한다.
  - 다른 트랜잭션이 데이터에 LS(X)을 걸어둔 경우, LS(X)의 요청은 허용하고 LX(X)는 허용하지 않는다.
  - 다른 트랜잭션이 데이터에 LX(X)을 걸어둔 경우, LS(X)와 LX(X) 모두 허용하지 않는다.
  - 트랜잭션이 락을 허용받지 못하면 대기 상태가 된다.

표 8-4 **락 호환행렬**

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

2.2.3 2단계 락킹

- **락을 사용하면 갱신손실 문제를 해결할 수 있다. 하지만 락을 걸고 해제하는 시점에 제한을 두지 않으면 두 개의 트랜잭션이 동시에 실행될 때 데이터의 일관성이 깨질 수 있다. 이것을 방지하기 위하여 2단계 락킹(2 phase locking) 기법을 사용한다.**

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

- **[작업 설명] 두 개의 데이터에 두 개의 트랜잭션이 접근하여 갱신하는 작업**

2.2.3 2단계 락킹

- **[문제 발생] 락을 사용하되 2단계 락킹 기법을 사용하지 않을 경우**

트랜잭션T1	트랜잭션T2	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=2190 이므로 일관성 제약 조건에 위배됨 */

2.2.3 2단계 락킹

- [문제 해결] 2단계 락킹 기법을 사용할 경우

트랜잭션T1	트랜잭션T2	A, B 값

LX(A)
t1=read_item(A);
t1=t1-100;
A=write_item(t1);
		A=900
		B=1000

	LX(A) ...(대기상태)...	

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=2200 이므로 일관성 제약 조건을 지킴 */

2.4 데드락

- **두 개 이상의 트랜잭션이 각각 자신의 데이터에 대하여 락을 획득하고 상대방 데이터에 대하여 락을 요청하면 무한 대기 상태에 빠질 수 있다. 이러한 현상을 데드락(deadlock) 혹은 교착상태라고 한다.**

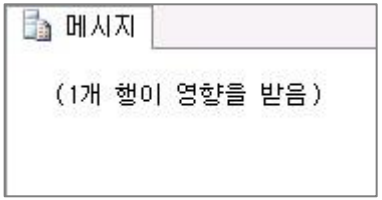
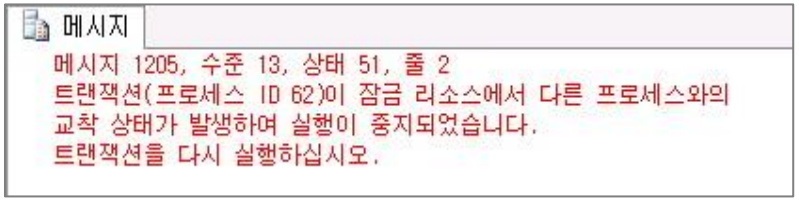
- **[작업 설명] 두 개의 데이터에 두 개의 트랜잭션이 접근하여 갱신하는 작업**

- **[문제 발생] SQL Server에서 데드락 발생**

- **[문제 해결] 데드락 해결**

  일반적으로 데드락이 발생하면 DBMS는 T1 혹은 T2의 작업 중 하나를 강제로 중지시킨다. 그 결과 나머지 트랜잭션은 정상적으로 실행된다. 이 때 중지시키는 트랜잭션에서 변경한 데이터는 원래 상태로 되돌려 놓는다.

2.4 데드락

트랜잭션 T1	트랜잭션 T2
BEGIN TRAN; USE Madang; UPDATE Book SET price=price+100 WHERE bookid=1;	
	BEGIN TRAN; USE Madang; 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; ...(대기상태)...
	

03. 트랜잭션 고립 수준

- 트랜잭션 동시 실행 문제
- 트랜잭션 고립 수준 명령어
- 트랜잭션 고립 수준 실습

3.1.1 오손 읽기

- 오손 읽기(dirty read)는 읽기 작업을 하는 트랜잭션 1이 쓰기 작업을 하는 트랜잭션 2가 작업한 중간 데이터를 읽기 때문에 생기는 문제다. 작업 중인 트랜잭션 2가 어떤 이유에서 작업을 철회(ROLLBACK)할 경우 트랜잭션 1은 무효가 된 데이터를 읽게 되고 잘못된 결과를 도출한다. 이 현상을 오손 읽기라고 한다.

- 오손 읽기 문제를 이해하기 위하여 다음과 같은 실습 테이블을 생성해보자.

```
/* 실습 테이블 생성 */
USE Madang;
Drop TABLE 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

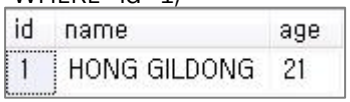
3.1.1 오손 읽기

[작업 설명] 두 개의 트랜잭션을 동시에 실행

트랜잭션 T1, T2가 동시에 실행된다. T1은 읽기만 하고 T2는 쓰기를 한다. T1은 T2가 변경한 데이터를 읽어와 작업하는데, T2가 작업 중 철회(ROLLBACK)를 하게 되었다.

[문제 발생] 오손 읽기

T2가 변경한 데이터를 T1이 읽은 후 어떤 원인으로 인하여 T2가 스스로 철회(ROLLBACK)를 하게 되었다. 철회를 하면 T2의 작업은 없던 일이 된다. T1은 T2가 종료하지 않은 상태에서 T2가 변경한 데이터를 보고 작업을 하게 된 것이다. 아래는 트랜잭션 T2가 홍길동의 나이를 30에서 21로 변경한 후 철회(ROLLBACK)하여, 트랜잭션 T1에게 오류를 발생시킨 예다.

T1(읽는 트랜잭션) READ UNCOMMITTED 모드	T2(쓰는 트랜잭션) READ COMMITTED 모드
BEGIN TRAN; USE Madang; SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED; SELECT * FROM Users WHERE id=1; 	
	BEGIN TRAN; USE Madang; UPDATE Users SET age=21 WHERE id=1; SELECT * FROM Users WHERE id=1; 
SELECT * FROM Users WHERE id=1; 	
	ROLLBACK
SELECT * FROM Users WHERE id=1; COMMIT; 	

3.1.2 반복불가능 읽기

- **반복불가능 읽기(non-repeatable read)는 트랜잭션 1이 데이터를 읽고 트랜잭션 2가 데이터를 쓰고(갱신, UPDATE) 트랜잭션 1이 다시 한 번 데이터를 읽을 때 생기는 문제다. 즉, 트랜잭션 1이 읽기 작업을 다시 한 번 반복할 경우 이전의 결과와 다른 결과가 나오는 현상을 반복불가능 읽기라고 한다.**

- **[작업 설명] 두 개의 트랜잭션을 동시에 실행**

  트랜잭션 T1, T2가 동시에 실행된다. T1은 읽기만 하고 T2는 쓰기(갱신, UPDATE)를 한다. T1은 데이터를 읽고 작업을 한 후, T2가 변경한 데이터를 다시 한 번 읽어와 작업을 한다.

- **[문제 발생] 반복불가능 읽기**

  T1이 데이터를 읽고 작업하던 중 T2가 데이터를 변경하였다. T1은 변경한 데이터를 보고 다시 한 번 작업을 하였다. 오손 읽기와 달리 이번에는 T2가 COMMIT을 했기 때문에 틀린 데이터는 아니다. 그런데 T1 입장에서는 같은 SQL 문이 다른 결과를 도출한다.

3.1.2 반복불가능 읽기

T1(읽는 트랜잭션) READ COMMITTED 모드	T2(쓰는 트랜잭션) READ COMMITTED 모드
BEGIN TRAN; USE Madang; SET TRANSACTION ISOLATION LEVEL READ COMMITTED; SELECT * FROM Users WHERE id=1; id | name | age 1 | HONG GILDONG | 30	
	BEGIN TRAN; USE Madang; 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	

3.1.3 유령데이터 읽기

- **유령데이터 읽기(phantom read)는 트랜잭션 1이 데이터를 읽고 트랜잭션 2가 데이터를 쓰고(삽입, INSERT) 트랜잭션 1이 다시 한 번 데이터를 읽을 때 생기는 문제다. 즉, 트랜잭션 1이 읽기 작업을 다시 한 번 반복할 경우 이전에 없던 데이터(유령 데이터)가 나타나는 현상을 유령데이터 읽기라고 한다.**

- **[작업 설명] 두 개의 트랜잭션을 동시에 실행**

  트랜잭션 T1은 읽기만 하고 T2는 쓰기(삽입, INSERT)를 한다. T1은 데이터를 읽고 작업을 한 후, T2가 변경한 데이터를 다시 한 번 읽어와 작업을 한다.

- **[문제 발생] 유령데이터 읽기**

  이번에는 T1이 T2가 새로운 데이터를 삽입한 사실을 모르고 작업을 한다. T2가 COMMIT을 했기 때문에 틀린 데이터는 아니다. 그러나 T1 입장에서는 새로운 데이터가 반영되어 반복불가능 읽기와 마찬가지로 같은 SQL 문이 다른 결과를 도출한다. 유령데이터 읽기는 반복불가능 읽기와 비슷하지만 없던 데이터가 삽입되었기 때문에 다르게 구분한다.

3.1.3 유령데이터 읽기

T1(읽는 트랜잭션) READ COMMITTED 모드	T2(쓰는 트랜잭션) READ COMMITTED 모드
BEGIN TRAN; USE Madang; SET TRANSACTION ISOLATION LEVEL REPEATABLE READ; SELECT * FROM Users WHERE age BETWEEN 10 AND 30; 	
	BEGIN TRAN; USE Madang; INSERT INTO Users VALUES (3, 'Bob', 27); COMMIT; SELECT * FROM Users WHERE age BETWEEN 10 AND 30; 
SELECT * FROM Users WHERE age BETWEEN 10 AND 30; COMMIT; 	

id	name	age
1	HONG GILDONG	21

id	name	age
1	HONG GILDONG	21
3	Bob	27

id	name	age
1	HONG GILDONG	21
3	Bob	27

3.2 트랜잭션 고립 수준 명령어

- **DBMS는 트랜잭션을 동시에 실행시키면서 락보다 좀 더 완화된 방법으로 문제를 해결하는 명령어를 제공하는데 이를 트랜잭션 고립 수준 명령어(transaction isolation level instruction)라고 한다.**

표 8-5 트랜잭션 고립 수준 명령어와 발생 현상

고립 수준 \ 문제	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

3.2.1 READ UNCOMMITTED(Level = 0)

- **READ UNCOMMITTED는 고립 수준이 가장 낮은 명령어로, 자신의 데이터에 아무런 공유락을 걸지 않는다(배타락은 갱신손실 문제때문에 걸어야 한다). 또한 다른 트랜잭션에 공유락과 배타락이 걸린 데이터를 대기하지 않고 읽는다. 심지어 다른 트랜잭션이 COMMIT하지 않은 데이터도 읽을 수 있다. 그 때문에 오손(dirty) 페이지의 데이터를 읽게 된다. 이 명령어는 SELECT 질의의 대상이 되는 테이블에 대해서 락을 설정하지 않은 것(NOLOCK)과 같다.**

표 8-6 READ UNCOMMITTED 모드 요약

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

3.2.2 READ COMMITTED(Level=1)

- **READ COMMITTED는 오손(dirty) 페이지의 참조를 피하기 위해 자신의 데이터를 읽는 동안 공유락을 걸지만 트랜잭션이 끝나기 전에라도 해지가능하다. 다른 트랜잭션 데이터는 락 호환성 규칙에 따라 진행한다. 이 옵션은 SQL Server의 기본 설정이다. 즉 아무런 설정을 하지 않으면 READ COMMITTED 방식으로 수행된다.**

표 8-7 READ COMMITTED 모드 요약

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

3.2.3 REPEATABLE READ(Level=2)

- 자신의 데이터에 설정된 공유락과 배타락을 트랜잭션이 종료할 때까지 유지하여 다른 트랜잭션이 자신의 데이터를 갱신(UPDATE)할 수 없도록 한다. 다른 트랜잭션 데이터는 락 호환성 규칙에 따라 진행한다. 다른 고립화 수준에 비해 데이터의 동시성(concurrency)이 낮아 특별하지 않은 상황이라면 사용하지 않는 것이 좋다.

표 8-8 REPEATABLE READ 모드 요약

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

3.2.4 SERIALIZABLE(Level=3)

- 고립 수준이 가장 높은 명령어로, 실행 중인 트랜잭션은 다른 트랜잭션으로부터 완벽하게 분리된다. 데이터 집합에 범위를 지어 잠금을 설정할 수 있기 때문에 다른 사용자가 데이터를 변경하려고 할 때 트랜잭션을 완벽하게 분리할 수 있다. 이 명령어는 네 가지 고립화 수준 중 제한이 가장 심하고 데이터의 동시성도 낮다. 이 명령어는 SELECT 질의의 대상이 되는 테이블에 미리 배타락을 설정한 것과 같은 효과를 낸다.

표 8-9 SERIALIZABLE 모드 요약

모드	SERIALIZABLE
LOCK	SELECT 문 - 공유락을 걸고 트랜잭션을 끝까지 유지 UPDATE 문 - 배타락 설정 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

3.3.1 반복불가능 읽기 문제와 방지를 위한 명령어

❶ 반복불가능 읽기 문제

트랜잭션 T1 READ COMMITTED 모드 (기본 모드)	트랜잭션 T2 READ COMMITTED 모드 (기본 모드)
BEGIN TRAN; USE Madang; SELECT SUM(price) 총액 FROM Book; 총액 144500	
	BEGIN TRAN; USE Madang; SELECT bookid, bookname, publisher, price FROM Book WHERE bookid=1; bookid | bookname | publisher | price 1 | 축구의 역사 | 굿스포츠 | 7000 SELECT SUM(price) 총액 FROM Book; 총액 144500 UPDATE Book SET price=price+500 WHERE bookid=1; SELECT SUM(price) 총액 FROM Book; COMMIT; 총액 145000
SELECT SUM(price) 총액 FROM Book; /* 앞의 결과와 다름 */ COMMIT; 총액 145000	

3.3.1 반복불가능 읽기 문제와 방지를 위한 명령어

❷ 반복불가능 읽기 문제를 방지하기 위한 명령어-REPEATABLE READ 모드

트랜잭션 T1 REPEATABLE READ 모드	트랜잭션 T2 READ COMMITTED 모드 (기본 모드)
BEGIN TRAN; USE Madang; SET TRANSACTION ISOLATION LEVEL REPEATABLE READ; SELECT SUM(price) 총액 FROM Book; 총액 144500	
	BEGIN TRAN; USE Madang; SELECT bookid, bookname, publisher, price FROM Book WHERE bookid=1; bookid | bookname | publisher | price 1 | 축구의 역사 | 굿스포츠 | 7000 SELECT SUM(price) 총액 FROM Book; 총액 144500 /* 여기까지 실행해본 후 진행 */
	UPDATE Book SET price=price+500 WHERE bookid=1; (쿼리를 실행하는 중 ...) /* 대기 상태가 됨, T1이 COMMIT하면 실행됨 */
SELECT SUM(price) 총액 FROM Book; /* 앞의 결과와 같음 */ COMMIT; 총액 144500	메시지 (1개 행이 영향을 받음)
	SELECT SUM(price) 총액 FROM Book; COMMIT; 총액 145000

3.3.2 유령데이터 읽기 문제와 방지를 위한 명령어

❶ 유령데이터 읽기 문제

트랜잭션 T1 REPEATABLE READ 모드	트랜잭션 T2 READ COMMITTED 모드 (기본 모드)
BEGIN TRAN; USE Madang; SET TRANSACTION ISOLATION LEVEL REPEATABLE READ; SELECT SUM(price) 총액 FROM Book; 총액 144500	
	BEGIN TRAN; USE Madang; SELECT SUM(price) 총액 FROM Book; 총액 144500 INSERT INTO Book VALUES (11, '테스트', '테스트출판사', 5500); SELECT SUM(price) 총액 FROM Book; COMMIT; 총액 150000
SELECT SUM(price) 총액 FROM Book; /* 앞의 결과와 다름 */ COMMIT; 총액 150000	

3.3.2 유령데이터 읽기 문제와 방지를 위한 명령어

❷ 유령데이터 읽기 문제를 방지하기 위한 명령어 - SERIALIZABLE 모드

트랜잭션 T1 SERIALIZABLE 모드	트랜잭션 T2 READ COMMITTED 모드 (기본 모드)
BEGIN TRAN; USE Madang; SET TRANSACTION ISOLATION LEVEL SERIALIZABLE; SELECT SUM(price) 총액 FROM Book; 총액 144500	
	BEGIN TRAN; USE Madang; SELECT SUM(price) 총액 FROM Book; /* 여기까지 실행해본 후 진행 */ 총액 144500
	INSERT INTO Book VALUES (11, '테스트', '테스트출판사',5500); (쿼리를 실행하는 중 ...) /* 대기 상태가 됨, T1이 COMMIT하면 실행됨 */
SELECT SUM(price) 총액 FROM Book; /* 앞의 결과와 같음 */ COMMIT; 총액 144500	메시지 (1개 행이 영향을 받음)
	SELECT SUM(price) 총액 FROM Book; COMMIT; 총액 150000

04. 회복

- 트랜잭션과 회복
- 로그 파일
- 로그 파일을 이용한 회복
- 체크포인트를 이용한 회복

04. 회복

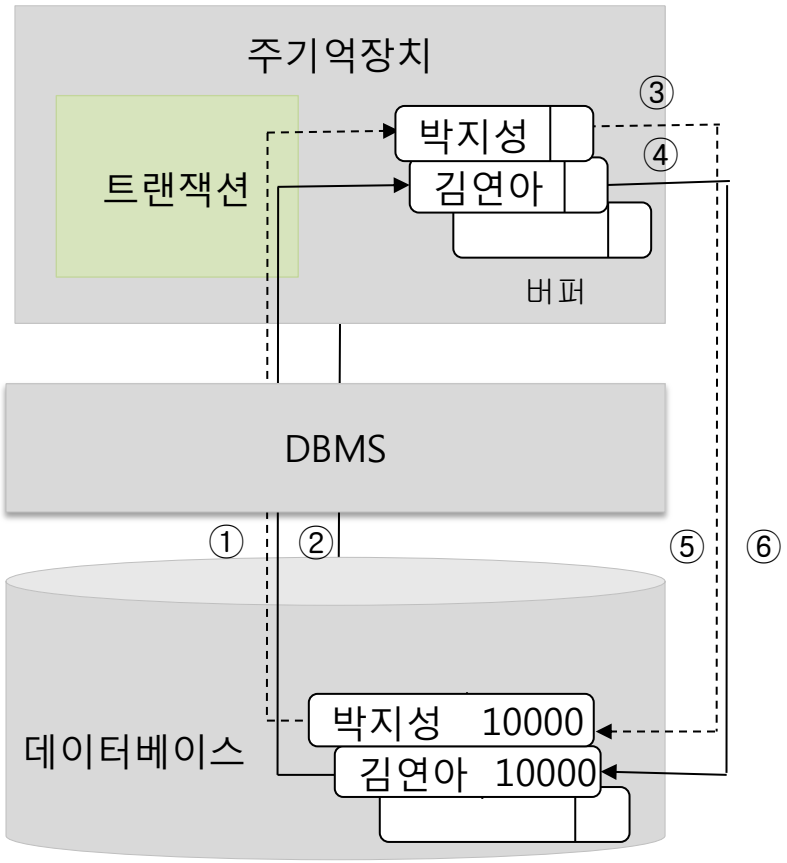
- **회복(recovery)은 데이터베이스에 장애가 발생했을 때 데이터베이스를 일관성 있는 상태로 되돌리는 DBMS의 기능이다.**

- **데이터베이스 시스템에서 발생할 수 있는 장애의 유형은 다음과 같다.**
  - 시스템 충돌 : 하드웨어 혹은 소프트웨어의 오류로 인하여 주기억장치가 손실되는 것을 말한다. 주기억장치에 상주하여 처리 중인 프로그램과 데이터의 일부 혹은 전부가 손실된다.
  - 미디어 장애 : 헤드의 충돌이나 읽기 장애에 의하여 보조기억장치의 일부 데이터가 손실되는 것을 말한다. 보조기억장치에 저장 중인 데이터의 일부 혹은 전부가 손실된다.
  - 응용 소프트웨어 오류 : 데이터베이스에 접근하는 소프트웨어의 논리적인 오류로 트랜잭션의 수행이 실패하는 것을 말한다.
  - 자연재해 : 화재, 홍수, 지진, 정전 등에 의해 컴퓨터 시스템이 손상되는 것을 말한다.
  - 부주의 혹은 태업sabotage : 운영자나 사용자의 부주의로 데이터가 손실되거나 의도적인 손상을입는 것을 말한다.

4.1 트랜잭션과 회복


```
BEGIN TRANSACTION
① /* 박지성 계좌를 읽어온다 */
② /* 김연아 계좌를 읽어온다 */
   /* 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 /* 부분완료 */
⑤ /* 박지성 계좌를 기록한다 */
⑥ /* 김연아 계좌를 기록한다 */

COMMIT TRANSACTION
```

(a) 계좌이체 트랜잭션



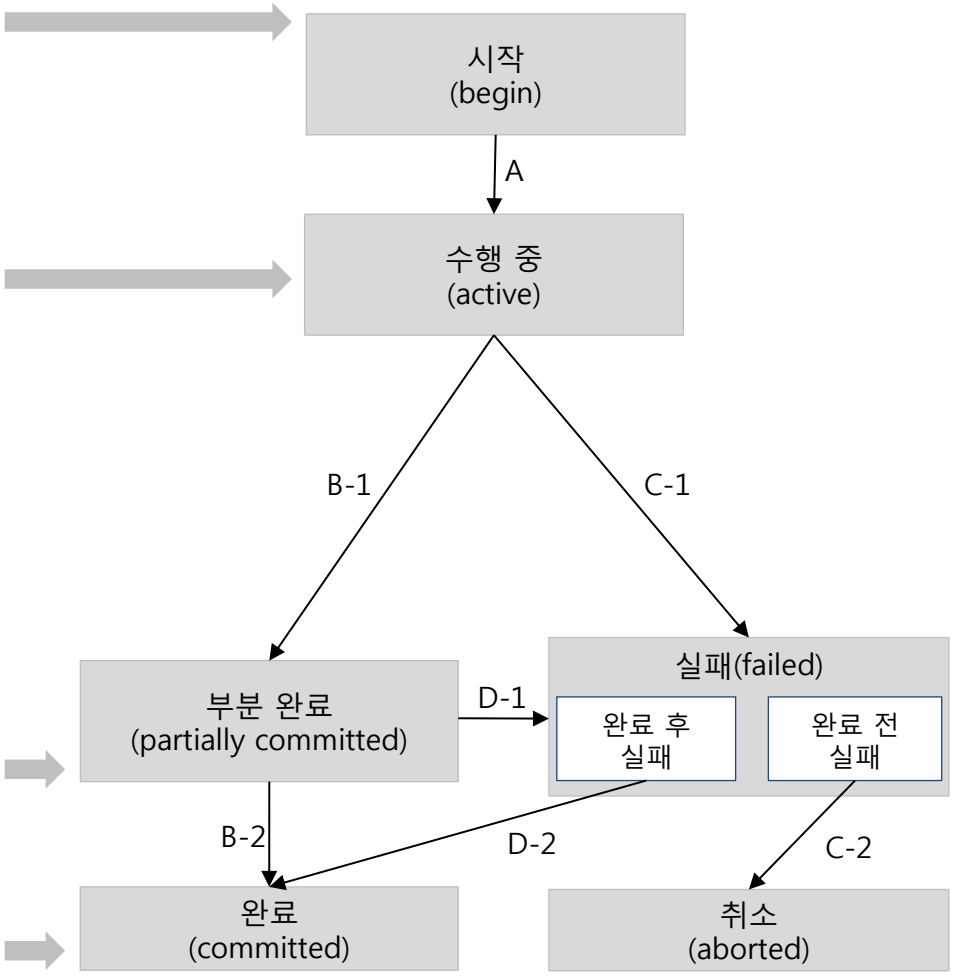
(b) 트랜잭션 수행과정

그림 8-11 **계좌이체 트랜잭션과 수행 과정**

4.1 트랜잭션과 회복


```
BEGIN TRANSACTION
① /* 박지성 계좌를 읽어온다 */
② /* 김연아 계좌를 읽어온다 */
   /* 잔고 확인 */

③ /* 예금인출 박지성 */
UPDATE Customer
SET balance=balance-100
WHERE name='박지성';

④ /* 예금입금 김연아 */
UPDATE Customer
SET balance=balance+100
WHERE name='김연아';

COMMIT /* 부분완료 */
⑤ /* 박지성 계좌를 기록한다 */
⑥ /* 김연아 계좌를 기록한다 */

COMMIT TRANSACTION
```

(a) 계좌이체 트랜잭션



(b) 트랜잭션 상태도

그림 8-12 **트랜잭션 수행과 상태도**

4.2 로그 파일

- **DBMS는 트랜잭션이 수행 중이거나 수행이 종료된 후 발생하는 데이터베이스 손실을 방지하기 위해 트랜잭션의 데이터베이스 기록을 추적하는 로그 파일(log file)을 사용한다. 로그 파일은 트랜잭션이 반영한 모든 데이터의 변경사항을 데이터베이스에 기록하기 전에 미리 기록해두는 별도의 데이터베이스다. 안전한 하드디스크에 저장되며 전원과 관계없이 기록이 남아있다.**

- **로그 파일에 저장된 로그의 구조는 다음과 같다.**

  <트랜잭션번호, 로그의 타입, 데이터 항목 이름, 수정 전 값, 수정 후 값>

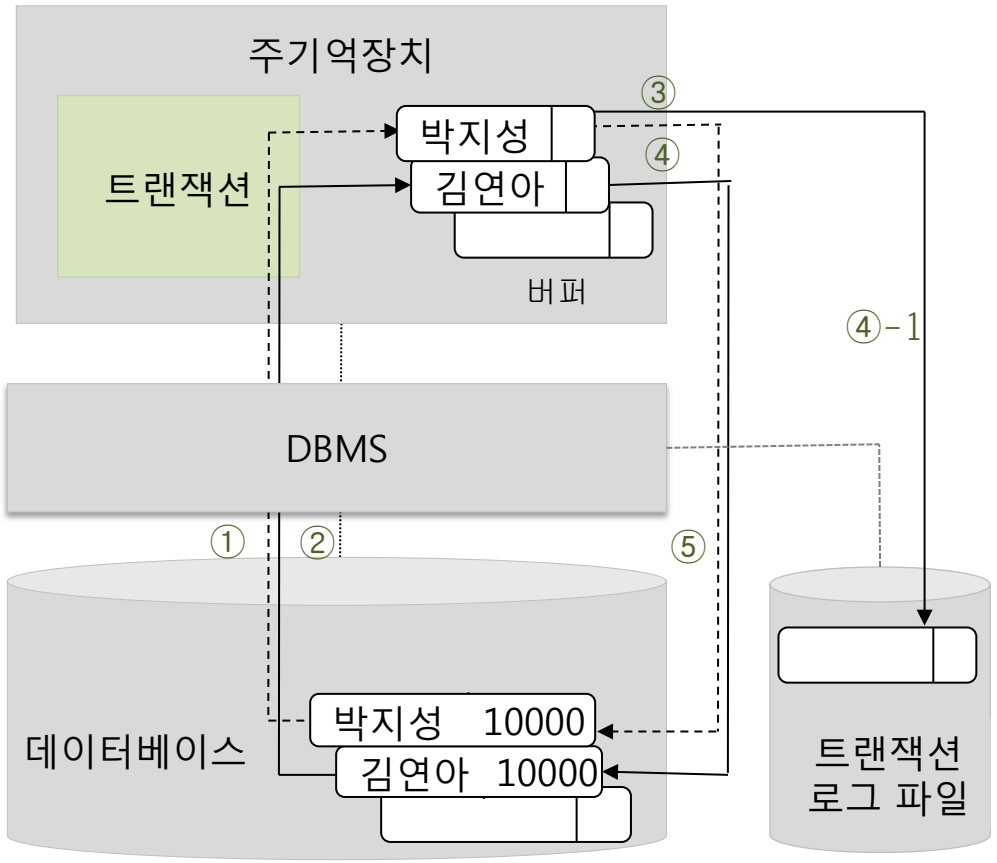
- **'로그의 타입'은 트랜잭션의 연산 타입으로 START, INSERT, UPDATE, DELETE, ABORT, COMMIT 등이 있다. '수정 전 값'은 데이터의 변경 전 값을 나타내고, '수정 후 값'은 연산의 결과로 변경된 값을 나타낸다.**

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

4.2 로그 파일


```
BEGIN TRANSACTION
① /* 박지성 계좌를 읽어온다 */
② /* 김연아 계좌를 읽어온다 */
   /* 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 /* 부분완료 */
⑤ /* 박지성 계좌를 기록한다 */
⑥ /* 김연아 계좌를 기록한다 */

COMMIT TRANSACTION
```

(a) 계좌이체 트랜잭션



(b) 트랜잭션 수행과정

그림 8-13 **트랜잭션 수행과 로그 파일**

4.3 로그 파일을 이용한 회복

- **데이터의 변경 기록을 저장해 둔 로그 파일을 이용하면 시스템 장애도 복구할 수 있다.**

  - 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);

4.3 로그 파일을 이용한 회복

- 트랜잭션이 T1 → T2 순으로 실행된다면 다음과 같은 로그 파일이 생성된다.

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]

4.3 로그 파일을 이용한 회복

- **시스템 운영 중 장애가 발생하여 시스템이 다시 가동되었을 때 DBMS는 로그 파일을 먼저 살펴본다. 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 재실행(REDO)**

  장애가 발생한 후 시스템을 다시 가동을 했을 때, 로그 파일에 트랜잭션의 시작(START)이있고 종료(COMMIT)가 있는 경우다. COMMIT 연산이 로그에 있다는 것은 트랜잭션이 모두 완료되었다는 의미다. 다만 변경 내용이 버퍼에서 데이터베이스에 기록되지 않았을 가능성이 있다. 따라서 로그를 보면서 트랜잭션이 변경한 내용을 데이터베이스에 다시 기록하는 과정이 필요하다. 이 과정을 REDO라고 한다.

- **트랜잭션의 취소(UNDO)**

  장애가 발생한 후 시스템을 다시 가동했을 때, 로그 파일에 트랜잭션의 시작(START)만 있고 종료(COMMIT)가 없는 경우다. COMMIT 연산이 로그에 보이지 않는다는 것은 트랜잭션이 완료되지 못했다는 의미로, 트랜잭션이 한 일을 모두 취소해야 한다. 이 경우 완료하지 못했지만 버퍼의 변경 내용이 데이터베이스에 기록되어 있을 가능성이 있기 때문에 로그를 보면서 트랜잭션이 변경한 내용을 데이터베이스에서 원상복구시켜야 한다. 이 과정을 UNDO라고 한다.

4.3 로그 파일을 이용한 회복

- **즉시 갱신 방법**

  즉시 갱신(immediate update)은 '갱신 데이터→로그', '버퍼→데이터베이스' 작업이 부분완료 전에 동시에 진행될 수 있으며, 부분완료가 되면 갱신 데이터는 로그에 기록이 끝난 상태다.

- **지연 갱신 방법**

  지연 갱신(deferred update)은 '갱신 데이터→로그'가 끝난 후 부분완료를 하고 '버퍼→데이터베이스' 작업이 진행되는 방법이다.

4.3 로그 파일을 이용한 회복

표 8-10 **트랜잭션 로그와 회복 방법(즉시 갱신 방법)**

로그 번호	작업	결과
i =0	아무 작업도 필요 없음	T1과 T2가 수행을 시작하지 않았음
1 < = i < = 4	UNDO(T1) : T1을 취소 → T1이 i까지 생성한 로그 레코드를 이용하여 데이터베이스 항목을 되돌림	T1을 수행하지 않은 것과 같음
5 < = i < = 9	REDO(T1) : T1을 재수행 → 1부터 4까지 T1이 생성한 로그 레코드를 이용하여 데이터베이스 항목 값을 기록함 UNDO(T2) : T2를 취소 → T2가 5부터 i까지 생성한 로그 레코드를 이용하여 데이터베이스 항목을 되돌림	T1은 수행이 완료됨 T2는 수행하지 않은 것과 같음
10	REDO(T1) : T1을 재수행 REDO(T2) : T2를 재수행	T1, T2는 수행이 완료됨

4.3 로그 파일을 이용한 회복

표 8-11 **트랜잭션 로그와 회복 방법(지연 갱신 방법)**

로그 번호	작업	결과
i =0	아무 작업도 필요 없음	T1과 T2가 수행을 시작하지 않았음
1 < = i < = 4	T1 : 아무 작업도 필요 없음	T1을 수행하지 않은 것과 같음
5 < = i < = 9	REDO(T1) : T1을 재수행 → 1부터 4까지 T1이 생성한 로그 레코드를 이용하여 데이터베이스 항목 값을 기록함 T2 : 아무 작업도 필요 없음	T1은 수행이 완료됨 T2는 수행하지 않은 것과 같음
10	REDO(T1) : T1을 재수행 REDO(T2) : T2를 재수행	T1, T2는 수행이 완료됨

4.4 체크포인트를 이용한 회복

- **로그를 이용한 회복은 시스템에 장애가 일어났을 때 어느 시점까지 되돌아가야하는지 알 수 없다. 트랜잭션이 많은 응용의 경우 하루 이상 되돌아가서 복구하는 것은 사실상 불가능하다. 회복 시 많은 양의 로그를 검색하고 갱신하는 시간을 줄이기 위하여 몇 십 분 단위로 데이터베이스와 트랜잭션 로그 파일을 동기화한 후 동기화한 시점을 로그 파일에 기록해두는 방법 혹은 그 시점을 체크포인트(checkpoint, 혹은 검사점)라고 한다.**

- **체크포인트 시점에는 다음과 같은 작업을 진행한다.**
  - 주기억장치의 로그 레코드를 모두 하드디스크의 로그 파일에 저장한다.
  - 버퍼에 있는 변경된 내용을 하드디스크의 데이터베이스에 저장한다.
  - 체크포인트를 로그 파일에 표시한다.

4.4 체크포인트를 이용한 회복

- **체크포인트가 있으면 로그를 이용한 회복 기법은 좀 더 간단해진다.**

- **체크포인트 이전에 [COMMIT] 기록이 있는 경우**

  아무 작업이 필요 없다. 로그에 체크포인트가 나타나는 시점은 이미 변경 내용이 데이터베이스에 모두 기록된 후이기 때문이다.

- **체크포인트 이후에 [COMMIT] 기록이 있는 경우**

  REDO(T)를 진행한다. 체크포인트 이후에 변경 내용이 데이터베이스에 반영되지 않았으므로 REDO를 진행한다.

- **체크포인트 이후에 [COMMIT] 기록이 없는 경우**

  즉시 갱신 방법을 사용했다면 UNDO(T)를 진행한다. 버퍼의 내용이 반영됐을 수도 있기 때문에 원상복구시켜야 한다. 반면 지연 갱신 방법을 사용했다면 아무것도 할 필요가 없다. 지연 갱신 방법은 [COMMIT] 이전에는 버퍼의 내용을 데이터베이스에 반영하지 않기 때문이다.

4.4 체크포인트를 이용한 회복

- 즉시 갱신 방법을 사용했다면 T2, T3는 아무 작업이 필요 없고, T4, T5는 REDO, T1, T6는 UNDO가 필요하다. 지연 갱신 방법을 사용했다면 T2, T3는 아무 작업이 필요 없고, T4, T5는 REDO가 필요하다. T1, T6는 아무 작업이 필요 없다.

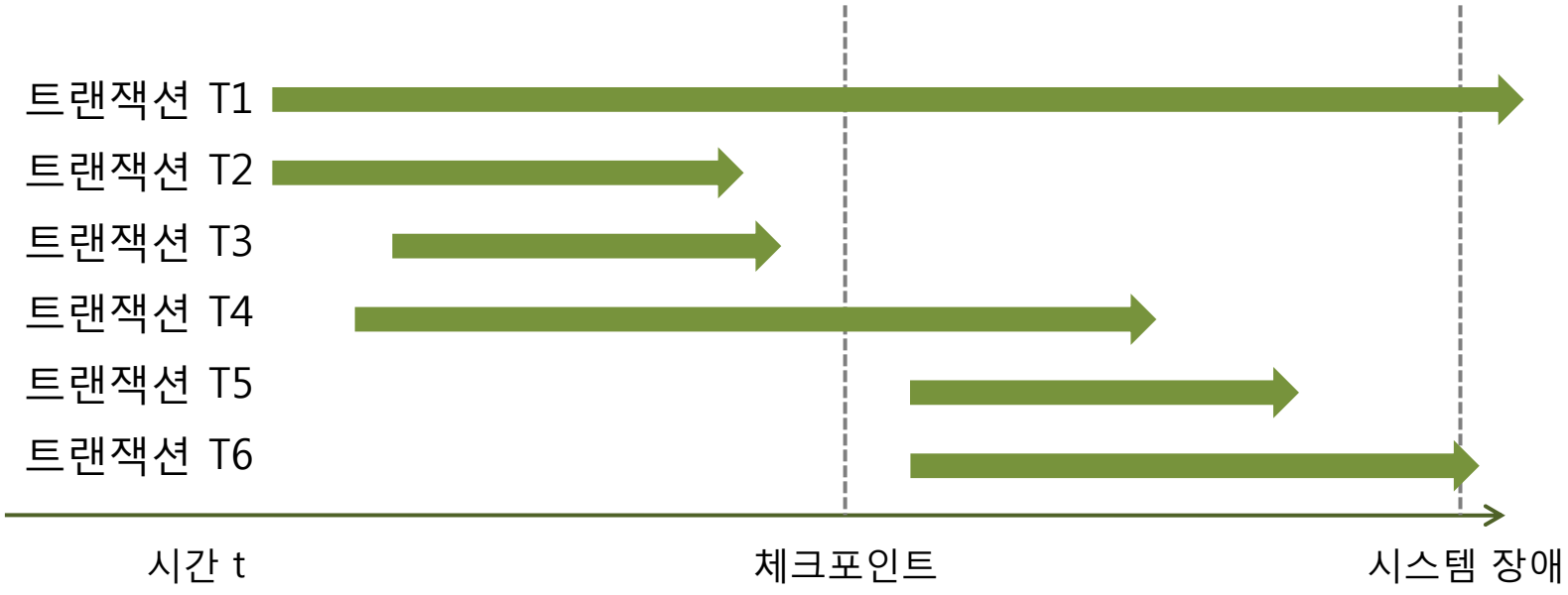


그림 8-15 트랜잭션 로그 기록과 체크포인트

4.4 체크포인트를 이용한 회복

- **트랜잭션 T1, T2, T3가 동시에 실행된 후 다음과 같이 로그 기록을 남겼다. 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

요약

1. 트랜잭션의 상태도
2. 트랜잭션의 성질
3. 동시성 제어
4. 갱신손실
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. 체크 포인트

SQL Server로 배우는 데이터베이스 개론과 실습