

12 장

트랜잭션과 병행수행 제어

Sejong Oh
Dankook University

목차

- 1. 트랜잭션
- 2. 장애와 회복
- 3. 병행 수행 제어
- 실습. 테이블을 Excel 파일로 저장

1. 트랜잭션

- 트랜잭션의 개념

- 트랜잭션(transaction) : 데이터베이스의 상태를 변화시키는 업무처리의 논리적 단위
- 트랜잭션은 현실세계의 업무를 반영
- Ex) 제품구매, 계좌이체, 수강신청
- 하나의 트랜잭션은 보통 여러 작업(연산)으로 구성

<A과목의 수강신청의 처리 과정>

- ① 홍길동의 수강과목에 A과목을 추가,
- ② A과목의 수강가능 인원을 1명 감소

두가지 연산이 모두 성공해야 트랜잭션이 완료됨

- 트랜잭션을 더 이상 쪼개면 업무의 의미를 상실하는 작업(연산)들의 집합으로 정의 가능

1. 트랜잭션

계좌

고객	계좌잔액
홍길동	10,000
김철수	10,000



계좌이체

```
update 계좌  
set 계좌잔액 = 계좌잔액 - 5000  
where 고객 = '홍길동'
```

```
update 계좌  
set 계좌잔액 = 계좌잔액 + 5000  
where 고객 = '김철수'
```



계좌

고객	계좌잔액
홍길동	5,000
김철수	15,000

(a) 계좌이체가 정상적으로 완료

계좌

고객	계좌잔액
홍길동	10,000
김철수	10,000



계좌이체

```
update 계좌  
set 계좌잔액 = 계좌잔액 - 5000  
where 고객 = '홍길동'
```



계좌

고객	계좌잔액
홍길동	5,000
김철수	10,000

(b) 계좌이체 과정에서 장애가 발생하여 비정상 종료

1. 트랜잭션

- 트랜잭션을 구성하는 작업들이 일부만 성공하면 문제 발생
- 트랜잭션을 구성하는 작업들은 '**모두 실행이 되든지, 하나도 실행이 안되든지 (all-or-nothing)**'해야 시스템에 문제를 일으키지 않는다.
- 데이터베이스의 입장에서 보면 트랜잭션은 작업수행에 필요한 SQL문들의 집합으로 정의
- 특히 트랜잭션은 데이터베이스의 상태를 변경하는 insert, update, delete문의 집합으로 구성
- (select문만으로 구성된 트랜잭션은 중간에 문제가 생겨도 데이터베이스의 상태를 변화시키지 않기 때문에 논의에서 제외)

1. 트랜잭션

- 트랜잭션의 4가지 특성

- 여러 개의 작업으로 구성된 트랜잭션은 수행시 원자성(atomicity), 일관성(consistency), 격리성(isolation), 지속성(durability) 이렇게 4가지의 특성이 지켜질 때 데이터베이스에 문제를 일으키지 않는다.

- 원자성(atomicity)

- 트랜잭션을 구성하는 작업들은 전부 수행이 되거나 전혀 수행이 되지 않아야 하는 특성
- '원자성'이라는 단어는 트랜잭션을 구성하는 작업들 전체를 하나로 보아야 한다는 의미

수강신청

```
insert 수강정보  
values ( ' 홍길동', '자료구조 '
```

```
update 개설과목  
set 수강가능인원 = 수강가능인원 -1  
where 과목명 = '자료구조 '
```

<그림 12-2> 계좌이체 트랜잭션의 원자성

1. 트랜잭션

● 일관성(consistency)

- 일관성이란 트랜잭션을 수행하기 이전의 데이터베이스와 수행한 이후의 데이터베이스가 논리적으로 일관된 상태를 유지하는 특성을 말한다

판매정보

상품명	판매수량
TV	0
냉장고	10

상품재고

상품명	재고수량
TV	20
냉장고	15

판매

```
update 판매정보  
set 판매수량 = 판매수량 + 5  
where 상품명 = 'TV'
```

```
update 상품재고  
Set 재고수량 = 재고수량 - 5  
where 상품명 = 'TV'
```

판매정보

상품명	판매수량
TV	5
냉장고	10

상품재고

상품명	재고수량
TV	15
냉장고	15

<그림 12-2> 판매 트랜잭션의 일관성

판매정보와 상품재고의 TV 수량 합계($0+20=20$)는 판매 트랜잭션이 실행된 후 ($5+15=20$)에도 동일하게 유지가 되어야 한다.

1. 트랜잭션

○ 격리성(isolation)

- 격리성이란 여러 트랜잭션들이 병렬적으로 수행되는 상황에서 트랜잭션들간에 상호 간섭에 의한 문제를 일으키지 않는 성질
- 격리성을 만족한다면 병렬 수행의 결과는 각 트랜잭션을 순차적으로 실행하였을 때의 결과와 같게 된다.
- 트랜잭션들의 병행 수행 상황에서 격리성을 만족시키려면 수행 중인 어떤 트랜잭션이 완료될 때까지 다른 트랜잭션들이 중간 연산 결과에 접근할 수 없도록 해야 한다.

계좌

고객	계좌잔액
홍길동	10,000
김철수	10,000

(수행 전)

계좌

고객	계좌잔액
홍길동	5,000
김철수	15,300

(수행 후)

계좌이체

```
select 계좌잔액 into A
from 계좌
where 고객 = '홍길동'
```

```
select 계좌잔액 into B
from 계좌
where 고객 = '김철수'
```

```
update 계좌
set 계좌잔액 = 계좌잔액 - 5000
where 고객 = '홍길동'
```

```
update 계좌
set 계좌잔액 = 계좌잔액 + 5000
where 고객 = '김철수'
```

이자지급

```
update 계좌
set 계좌잔액 = 계좌잔액 + 300
where 고객 = '김철수'
```



(a) 계좌이체, 이자지급 트랜잭션의 순차적 실행

계좌

고객	계좌잔액
홍길동	10,000
김철수	10,000

(수행 전)

계좌

고객	계좌잔액
홍길동	5,000
김철수	15,000

(수행 후)

계좌이체

```
select 계좌잔액 into A
from 계좌
where 고객 = '홍길동'
```

```
select 계좌잔액 into B
from 계좌
where 고객 = '김철수'
```

```
update 계좌
set 계좌잔액 = A - 5000
where 고객 = '홍길동'
```

```
update 계좌
set 계좌잔액 = B + 5000
where 고객 = '김철수'
```

이자지급

```
update 계좌
set 계좌잔액 = 계좌잔액 + 300
where 고객 = '김철수'
```

(b) 계좌이체, 이자지급 트랜잭션의 병렬적 실행

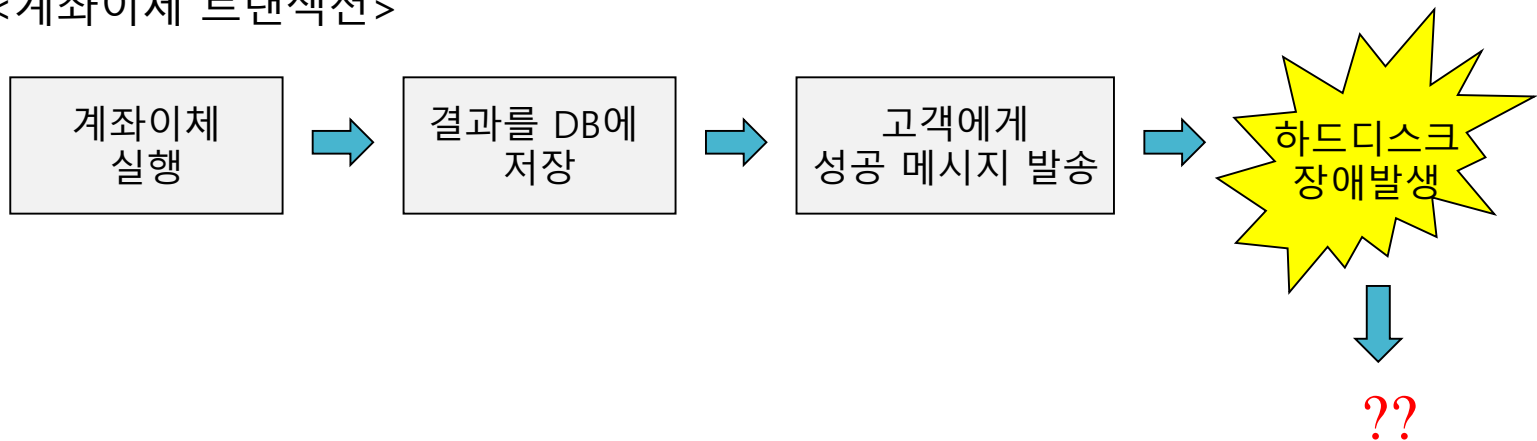
<그림 12-3> 병행 수행에서 격리성의 문제

1. 트랜잭션

○ 지속성(durability)

- 지속성이란 트랜잭션이 성공적으로 완료된 후에는 트랜잭션의 수행 결과가 데이터베이스에 영구적으로 유지되는 특성
- 트랜잭션이 성공적으로 완료되었다면 시스템에 장애가 발생하더라도 트랜잭션의 수행결과가 데이터베이스에 반드시 반영되도록 DBMS가 조치를 취해야함을 의미

<계좌이체 트랜잭션>



정상 실행된 계좌이체 결과는 어떻게 되는가

1. 트랜잭션

- commit과 rollback 연산

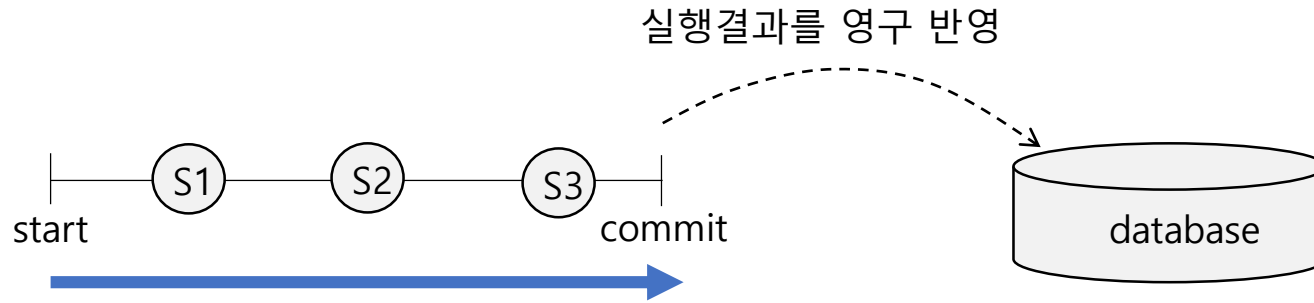
- 복습:

- 갱신 연산(insert, update, delete)을 수행한 다음에 commit을 실행해야 그 결과가 데이터베이스에 저장
 - commit 대신 rollback을 실행하면 갱신 연산의 실행 결과가 취소되어 데이터베이스에는 아무 변화가 발생하지 않는다.

- commit과 rollback 연산은 트랜잭션에도 적용

- commit은 트랜잭션이 성공적으로 수행되었음을 확정하는 명령어로서 commit 연산의 결과, 트랜잭션의 전체 수행결과가 데이터베이스에 영구적으로 기록
 - rollback은 트랜잭션의 처리 과정에서 발생한 전체 변경 사항을 취소하고, 트랜잭션 과정을 종료시키는 명령어로서, 트랜잭션의 수행이 어떤 이유로 실패했을 때 내리는 명령

1. 트랜잭션



(a) 트랜잭션이 정상 실행된 경우



(b) 트랜잭션의 실행 도중 장애가 발생한 경우

<그림 12-4> commit과 rollback

1. 트랜잭션

- 하나의 트랜잭션은 commit을 만나서 실행결과를 데이터베이스에 영구 반영하거나, rollback을 만나서 트랜잭션 시작 이전으로 되돌아오거나 둘중 하나로 마무리된다.
- commit에 의해서 데이터베이스에 저장된 결과는 rollback으로 취소할 수 없다



commit과 rollback은 트랜잭션의 특성중 원자성을 보장할 수 있는 방법입니다.

1. 트랜잭션

DDL, DCL과 commit

- SQL문중 DDL에 해당하는 create, alter, drop과 DCL에 해당하는 grant, revoke 데이터베이스에 변화를 일으키는 연산
- 따라서 이러한 명령문을 실행한 후에도 commit을 실행해야 데이터베이스에 반영이 될 것 같지만 DDL, DCL은 commit을 필요로 하지 않는다.
- 그 이유는 DDL, DCL은 실행 즉시 자동적으로 commit이 이루어져 그 결과가 데이터베이스에 반영되기 때문이다 (이를 auto commit이라고 한다).
- select문도 commit을 필요로 하지 않는다. 데이터베이스의 내용을 단순히 조회하는 연산은 데이터를 변화시키지 않기 때문이다,

1. 트랜잭션

- **오라클에서의 트랜잭션**

- 오라클도 트랜잭션의 개념을 지원
- 명시적으로 트랜잭션의 시작을 표시하는 명령어는 없으며 다음의 상황에서 트랜잭션이 시작

① 새로운 세션이 시작된 직후

SQL Plus나 SQL Developer에서 데이터베이스에 로그인하는 순간 새로운 세션이 시작된다. 또는 응용 프로그램에서 데이터베이스에 접속하여도 새로운 세션이 시작.

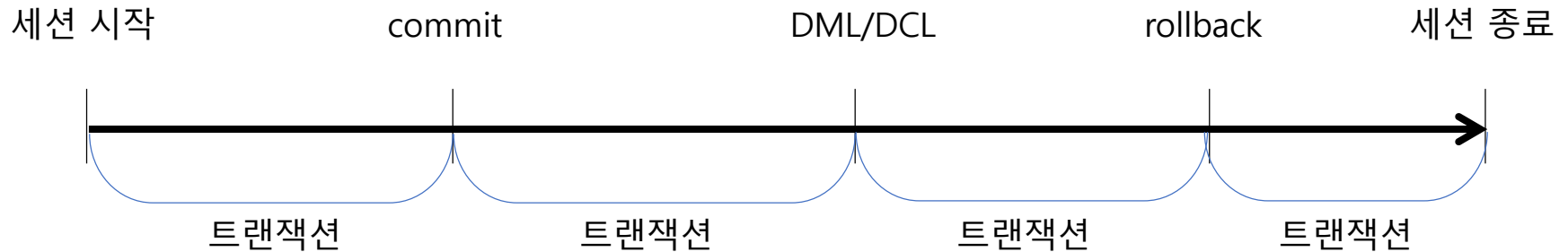
② commit, rollback 명령어를 실행한 직후

commit, rollback 명령어를 실행하면 하나의 트랜잭션이 끝나는 동시에 새로운 트랜잭션이 시작.

③ DDL, DCL 명령어를 실행한 직후

DDL, DCL을 실행하면 자동적으로 commit이 이루어지기 때문에 (auto commit) 트랜잭션이 종료 되고 새로운 트랜잭션이 시작

1. 트랜잭션



<그림 12-5> 시간의 흐름에 따른 트랜잭션의 시작과 종료

<Rollback 으로 끝나는 트랜잭션>

1	SELECT * FROM dept ;	} 트랜잭션
2	INSERT INTO dept VALUES (70, 'BRANCH_1', 'SEOUL') ;	
3	SELECT * FROM dept ;	
4	ROLLBACK ;	
5	SELECT * FROM dept ;	

1. 트랜잭션

DEP...	DNAME	LOC
1	10 ACCOUNTING	NEW YORK
2	20 RESEARCH	DALLAS
3	30 SALES	CHICAGO
4	40 OPERATIONS	BOSTON

<LINE 1>

DEPTNO	DNAME	LOC
1	10 ACCOUNTING	NEW YORK
2	20 RESEARCH	DALLAS
3	30 SALES	CHICAGO
4	40 OPERATIONS	BOSTON
5	70 BRANCH_1	SEOUL

<LINE 2,3> INSERT 실행됨

DEP...	DNAME	LOC
1	10 ACCOUNTING	NEW YORK
2	20 RESEARCH	DALLAS
3	30 SALES	CHICAGO
4	40 OPERATIONS	BOSTON

<LINE 4,5> ROLLBACK 실행 후

1. 트랜잭션

<Commit으로 종료되는 트랜잭션>

```
1 SELECT * FROM dept ;  
2 INSERT INTO dept VALUES (70, 'BRANCH_1', 'SEOUL') ;  
3 COMMIT;  
4 SELECT * FROM dept ;  
5 ROLLBACK ;  
6 SELECT * FROM dept ;
```

트랜잭션

트랜잭션

1. 트랜잭션

	DEPTNO	DNAME	LOC
1	10	ACCOUNTING	NEW YORK
2	20	RESEARCH	DALLAS
3	30	SALES	CHICAGO
4	40	OPERATIONS	BOSTON

<LINE 1>

	DEPTNO	DNAME	LOC
1	10	ACCOUNTING	NEW YORK
2	20	RESEARCH	DALLAS
3	30	SALES	CHICAGO
4	40	OPERATIONS	BOSTON
5	70	BRANCH_1	SEOUL

<LINE 2,3,4> INSERT 실행됨, COMMIT 실행됨

	DEPTNO	DNAME	LOC
1	10	ACCOUNTING	NEW YORK
2	20	RESEARCH	DALLAS
3	30	SALES	CHICAGO
4	40	OPERATIONS	BOSTON
5	70	BRANCH_1	SEOUL

<LINE 5,6> ROLLBACK 실행

이미 COMMIT 이 실행되었기 때문에
입력 결과가 취소되지 않는다

1. 트랜잭션

<DCL을 포함하는 트랜잭션의 실행>

```
1 SELECT * FROM dept ;
2 INSERT INTO dept VALUES (80, 'BRANCH_2', 'BUSAN') ;
3 GRANT update ON DEPT TO salesman_1 ;
4 SELECT * FROM dept ;
5 ROLLBACK ;
6 SELECT * FROM dept ;
```

트랜잭션

트랜잭션

1. 트랜잭션

	DEPTNO	DNAME	LOC
1	10	ACCOUNTING	NEW YORK
2	20	RESEARCH	DALLAS
3	30	SALES	CHICAGO
4	40	OPERATIONS	BOSTON
5	70	BRANCH_1	SEOUL

<LINE 1>

	DEPTNO	DNAME	LOC
1	10	ACCOUNTING	NEW YORK
2	20	RESEARCH	DALLAS
3	30	SALES	CHICAGO
4	40	OPERATIONS	BOSTON
5	70	BRANCH_1	SEOUL
6	80	BRANCH_2	BUSAN

<LINE 2,3> INSERT 실행됨, COMMIT 실행됨

	DEPTNO	DNAME	LOC
1	10	ACCOUNTING	NEW YORK
2	20	RESEARCH	DALLAS
3	30	SALES	CHICAGO
4	40	OPERATIONS	BOSTON
5	70	BRANCH_1	SEOUL
6	80	BRANCH_2	BUSAN

<LINE 4,5,6> ROLLBACK 실행

GRANT에 의해 COMMIT 이
실행되었기 때문에
입력 결과가 취소되지 않는다

1. 트랜잭션

Note. commit, rollback 명령은 누가 내리는가

- commit은 트랜잭션이 완료되었음을 확정하는 것이므로 어디서부터 어디까지가 트랜잭션인지를 알고 있는 사용자에게 의해서 명령이 내려진다.
- rollback은 사용자의 판단에 의해 명시적으로 내려지는 경우도 있지만, 장애가 발생한 후에 데이터베이스를 복구하는 과정에서 DNBMS의 판단으로 자동 실행이 되는 경우가 더 많다.



commit 명령에 의해 데이터베이스에 영구 반영된 결과는 rollback 연산으로 취소할 수 없습니다

2. 장애와 회복

● 개요

- 데이터베이스 시스템도 소프트웨어와 하드웨어로 구성된 시스템이기 때문에 다양한 장애(failure)가 발생
- 장애의 발생에 의해 데이터베이스가 논리적 일관성을 상실한 상태가 된다면 데이터베이스를 기반으로 이루어지는 현실 업무에도 영향
- 손상된 데이터베이스를 원래의 일관된 상태로 회복(recovery)할 수 있는 방법이 필요

<표 12-1> 데이터베이스 장애의 종류와 원인

유형	의미와 원인	
트랜잭션 장애	의미	트랜잭션을 더 이상 수행할 수 없는 상태가 됨
	원인	트랜잭션 자체의 논리적 오류, DBMS에 의한 수행 중단 (시스템 자원의 과다요구, 트랜잭션간 교착상태(dead lock) 발생 등)
시스템 장애	의미	DBMS가 정상적으로 작동될 수 없는 상태가 됨
	원인	하드웨어 또는 소프트웨어의 이상 발생 (메인 메모리 이상, 운영체제 에러, 전원공급 이상 등)
미디어 장애	의미	저장장치와 SW간 데이터 입출력이 정상적으로 이루어지지 못함
	원인	디스크 손상, 불량섹터 발생, 디스크 헤드 이상 등

2. 장애와 회복

- 백업(backup)과 로그(log)

- 데이터베이스에 장애가 발생했을 때, 데이터베이스를 원래의 일관된 상태로 회복하기 위한 기본적인 방법은 백업(backup)과 로그(log) 데이터를 이용하는 것
- **백업**은 운영중인 데이터베이스의 내용을 안전한 장소에 복사하여 보관하는 행위

<표 12-2> 시스템 종료가 필요한지 여부에 따른 백업의 분류

종류	설명
콜드 백업 (cold backup)	실행중인 운영 시스템을 중지한 후에 백업을 실시한다. 간편하나 대다수의 운영 시스템이 24시간 운영되기 때문에 백업 시간을 찾기 어려울 수 있다.
핫 백업 (hot backup)	운영 시스템을 중지하지 않고 백업을 수행한다. 백업을 하는 중에도 데이터베이스가 변화하기 때문에 백업된 데이터를 이용하여 회복작업을 할 때 주의가 필요하다.
웜 백업 (warm backup)	위 두방법을 조합한 형태로 운영 시스템을 중지하지는 않으나, 외부사용자가 데이터베이스를 변경하는 것을 막아서 운영시스템을 중지하고 백업하는 것과 유사한 효과를 얻을 수 있다.

2. 장애와 회복

<표 12-3> 백업의 범위에 따른 분류

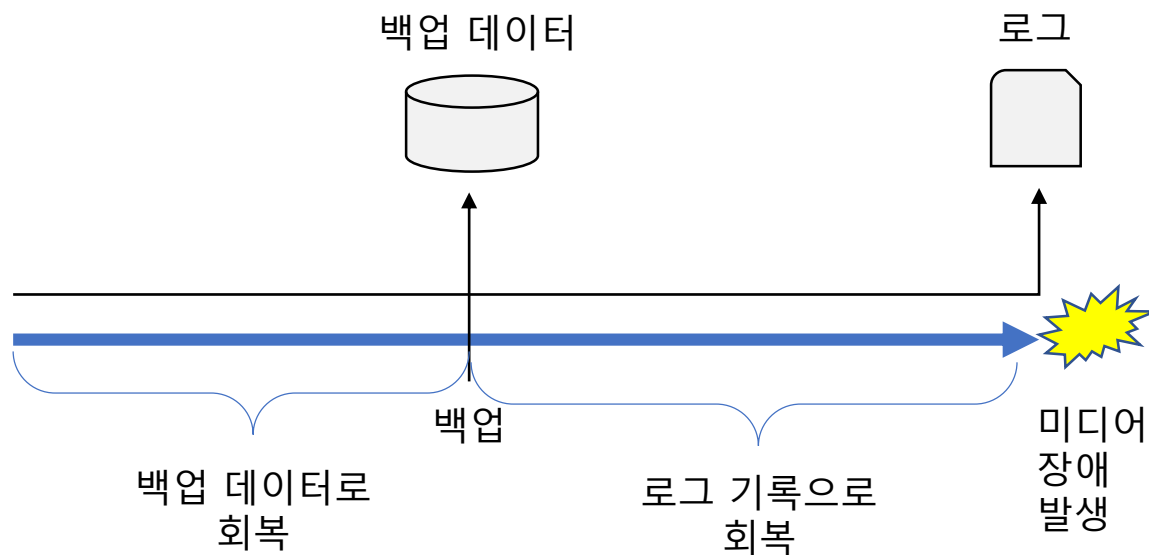
종류	설명
전체 백업 (full backup)	시스템 내의 백업 대상이 되는 모든 데이터를 백업하는 방식을 말한다. 백업에 시간이 오래 걸린다
증분 백업 (incremental backup)	과거의 백업 시점 이후에 변화된 부분만 백업하는 방식이다.

- 데이터베이스의 백업은 일정 주기로 이루어지기 때문에 백업 이후 시간이 지날수록 실제 데이터베이스와 백업 데이터는 차이가 생길 수밖에 없다.
- 백업 데이터만 가지고 회복을 한다면 백업 이후에 변화된 부분은 회복할 방법이 없음

2. 장애와 회복

로그

- 데이터베이스에 변화가 생길 때마다 시간과 함께 변화된 내용을 기록한 것
- 특정 데이터에 변화가 생기면 변화되기 이전 상태와 이후 상태를 함께 기록
- 보안 침해 사고가 발생했을 때 원인을 규명할 때도 중요한 참조 자료
- 데이터베이스에 장애가 발생 했을 때 회복을 위한 자료로 활용



<그림 12-5> 백업과 로그를 이용한 회복의 개념

2. 장애와 회복

Note. 회복시 백업과 로그의 사용

- 데이터베이스에 장애가 발생하였을 때 디스크 손상 같은 미디어 장애가 아니라면 장애 발생 후에도 데이터베이스에 접근하는 것은 가능하다.
- 다만 데이터베이스의 논리적 일관성은 침해되었을 가능성이 높음. 이런 경우는 백업 데이터를 이용하지 않고 로그 기록만으로 회복 작업이 가능하다. 따라서 다음과 같이 정리할 수 있다.
 - ✓ 미디어 장애 발생시 : 백업 데이터 + 로그 기록으로 회복 작업 수행
 - ✓ 기타 장애 발생시 : 로그 기록으로 회복 작업 수행
- ※ 미디어 장애가 발생하는 것은 매우 드문 일이므로 데이터베이스의 장애 회복은 보통 로그를 이용한 회복을 의미한다.

2. 장애와 회복

- 로그 레코드
 - 로그는 레코드(record) 단위로 기록

<표 12-3> 로그 레코드의 예

로그 레코드	의미
<T ₁ , start>	트랜잭션 T ₁ 이 시작됨
<T ₁ , X, old_value, new_value>	트랜잭션 T ₁ 이 데이터 X의 이전값 old_vlaue를 new_value로 갱신함. (예) <T ₁ , X, 100, 200>
<T ₁ , commit>	트랜잭션 T ₁ 에 대해 commit 연산이 실행됨
<T ₁ , abort>	트랜잭션 T ₁ 이 철회됨
<check point>	버퍼에 있던 내용이 모두 물리적 데이터베이스에 반영됨

2. 장애와 회복

- 판매 트랜잭션의 수행과 로그 기록

판매 트랜잭션 T_1

```
read(X) ;  
X = X + 5 ;  
write (X) ;
```

```
read(Y) ;  
Y = Y - 5 ;  
write (Y) ;
```



로그 기록

```
1: <T1, start>  
2: <T1, X, 0, 5>  
3: <T1, Y, 20, 15>  
4: <T1, commit>
```

2. 장애와 회복

- redo와 undo

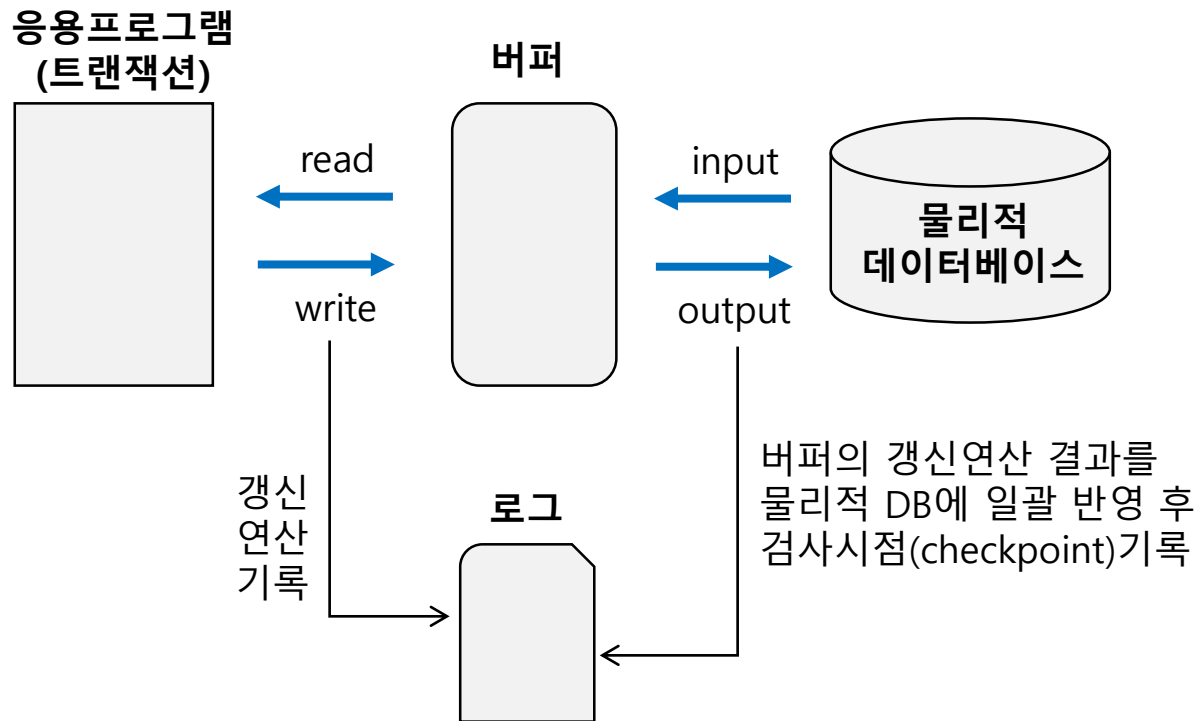
- 데이터베이스 장애 발생시, 백업 시점까지는 단순히 백업 데이터를 복사해 오는 것으로 복구가 가능
- 백업 시점 이후는 로그 기록을 이용하는데, 트랜잭션의 원자성을 유지하면서 회복을 해야하기 때문에 세심한 작업이 필요
- DBMS는 회복 작업시 로그에 기록된 트랜잭션에 대해 다음의 두가지 연산중 하나를 실행

redo (재실행)	<ul style="list-style-type: none">로그에 기록된 트랜잭션을 재실행한다.장애 발생 이전에 commit이 완료된 트랜잭션을 대상으로 한다.
undo (실행취소)	<ul style="list-style-type: none">로그에 기록된 트랜잭션을 취소한다. (트랜잭션의 실행으로 인한 데이터베이스의 변경된 내용을 변경 이전 상태로 되돌린다)장애 발생 시점에 실행중이던 트랜잭션(아직 commit이 실행되지 못한)을 대상으로 한다.

2. 장애와 회복

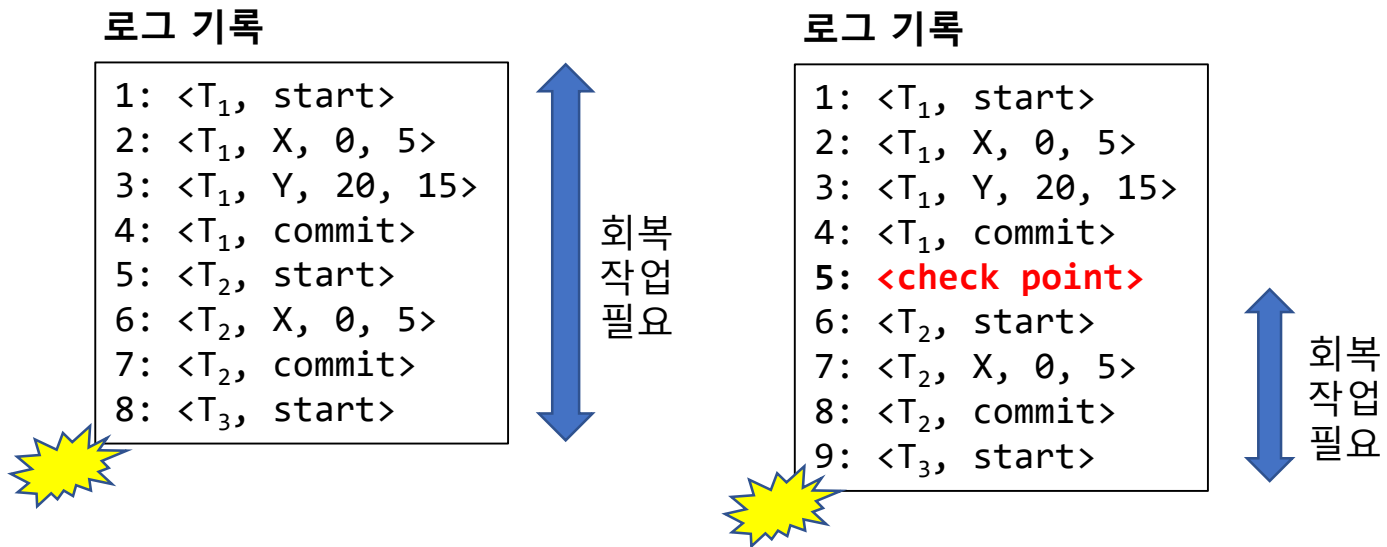
- 즉시 갱신과 지연 갱신

- 응용 프로그램 또는 트랜잭션에서 데이터베이스에 대해 읽거나 쓰기 연산을 수행하면 물리적 데이터베이스에 대해 직접 연산이 이루어지는 것이 아니라 **버퍼(buffer)**를 통해 연산이 이루어짐



2. 장애와 회복

- DBMS는 주기적으로 버퍼의 갱신된 데이터를 물리적 데이터베이스에 저장하여 로그의 내용과 물리적 데이터베이스의 내용을 일치시킨 후에 로그에 '**검사시점(checkpoint)**'을 기록
- 검사시점은 그 시점에 로그의 내용과 물리적 데이터베이스의 내용이 일치되었음을 의미

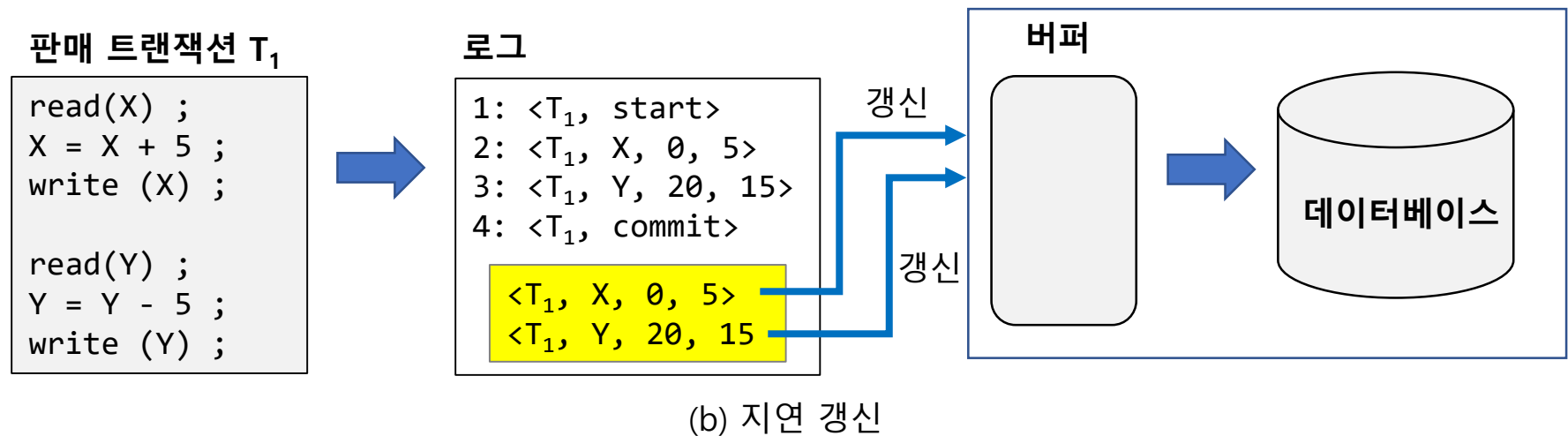
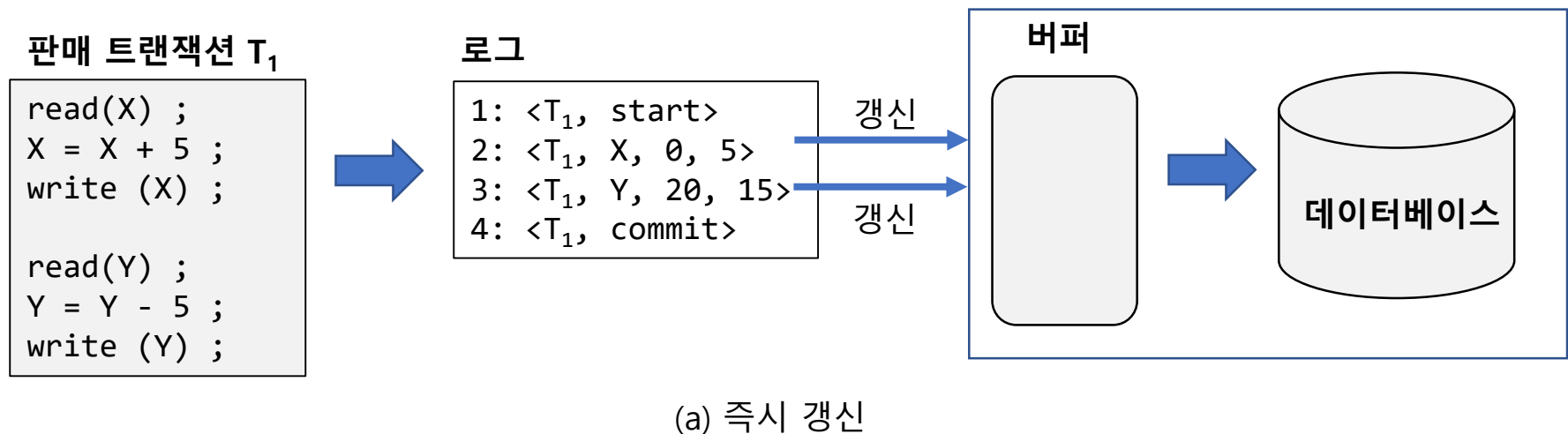


<그림> 체크 포인트와 회복작업

2. 장애와 회복

- 트랜잭션의 실행 내용을 반영하는 두가지 방법: 즉시 갱신과 지연 갱신
- 즉시 갱신(immediate update)**
 - 트랜잭션을 구성하는 연산들이 여러 개 있을 때 각 연산이 실행되는 즉시 그 결과를 데이터베이스(정확히는 버퍼)에 반영하는 방법
- 지연 갱신(deferred update)**
 - 트랜잭션을 구성하는 연산들이 실행되는 동안 로그에만 기록을 하고 있다가 commit 연산이 실행되면 그 때 각 연산의 결과를 한꺼번에 데이터베이스(정확히는 버퍼)에 반영하는 방법

2. 장애와 회복



<그림 12-8> 즉시 갱신과 지연 갱신의 사례

2. 장애와 회복

- ◉ Note. 트랜잭션의 실행을 데이터베이스와 로그에 기록할 때의 순서
 - 먼저 로그에 기록을 한 후 데이터베이스에 실행 결과를 반영
 - 데이터베이스에 먼저 결과를 저장한 후 로그에 기록하는 경우, 데이터베이스에 저장한 직후 장애가 발생하면 로그에는 기록이 안되어 있기 때문에 회복 작업시 해당 트랜잭션은 로그 기록에서 찾을 수가 없고, 따라서 그 트랜잭션은 복구가 불가능



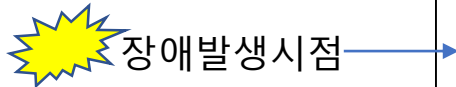
트랜잭션의 실행은 로그에 먼저 기록한 후에 데이터베이스에 실행 결과를 반영합니다.

2. 장애와 회복

- 로그 회복 기법
 - 예제 상황

로그

```
1 : <T1, start>  
2 : <T1, X, 0, 5>  
3 : <T1, Y, 20, 15>  
4 : <T1, commit>  
5 : <checkpoint>  
6 : <T2, start>  
7 : <T2, X, 5, 7>  
8 : <T2, Y, 15, 13>  
9 : <T2, commit>  
10: <T3, start>  
11: <T3, X, 7, 9>
```



2. 장애와 회복

● 즉시 갱신시 회복 기법

- 로그 기록을 보면 5번 시점에 마지막 검사시점(checkpoint)이 기록
- 트랜잭션 T_1 은 commit 연산까지 정상적으로 실행이 되었고 데이터베이스에도 정상적으로 반영되었으므로 아무 작업도 할 필요가 없다. (검사시점 이전에 트랜잭션 완료)
- 트랜잭션 T_2 은 commit 연산까지 정상적으로 실행이 되어 7번, 8번 시점의 연산 결과가 데이터베이스에 정상적으로 반영되었어야 하지만 버퍼만 갱신을 하고 아직 데이터베이스에는 반영이 안되었으므로 redo (재실행) 작업을 수행
- 트랜잭션 T_3 의 경우는 11번 시점의 연산이 수행되었고 즉시 갱신에 따라 버퍼에도 반영된 상태였다. 그러나 commit 연산이 수행되기 전에 장애가 발생하였기 때문에 트랜잭션의 원자성을 지키기 위해 **부분 수행된 트랜잭션 T_3 의 연산들은 취소가 되어야 한다.** 따라서 트랜잭션 T_3 에 대해서는 undo (실행취소) 작업을 수행

로그

```
1 : <T1, start>
2 : <T1, X, 0, 5>
3 : <T1, Y, 20, 15>
4 : <T1, commit>
5 : <checkpoint>
6 : <T2, start>
7 : <T2, X, 5, 7>
8 : <T2, Y, 15, 13>
9 : <T2, commit>
10: <T3, start>
11: <T3, X, 7, 9>
```

2. 장애와 회복

지연 갱신시 회복 기법

- 트랜잭션 T_1 은 commit 연산까지 정상적으로 실행이 되었고 데이터베이스에도 정상적으로 반영되었으므로 아무 작업도 할 필요가 없다
- 트랜잭션 T_2 은 commit 연산까지 정상적으로 실행이 되어 7번, 8번 시점의 연산 결과가 데이터베이스에 정상적으로 반영되었어야 하지만 버퍼만 갱신을 하고 아직 데이터베이스에는 반영이 안되었으므로 redo (재실행) 작업을 수행하여 데이터베이스에 반영
- 트랜잭션 T_3 의 경우는 11번 시점의 연산이 수행되었으나 지연 갱신 정책에 따라 로그에만 기록이 되고 버퍼에 반영이 안된 상태였다. 그리고 commit 연산이 수행되기 전에 장애가 발생하였기 때문에 **사실상 트랜잭션 T_3 의 연산들은 하나도 실행이 안된 것과 마찬가지**이다. 따라서 트랜잭션 T_3 에 대해서는 무시하고 아무 작업도 수행하지 않아도 된다.

로그

```
1 : <T1, start>
2 : <T1, X, 0, 5>
3 : <T1, Y, 20, 15>
4 : <T1, commit>
5 : <checkpoint>
6 : <T2, start>
7 : <T2, X, 5, 7>
8 : <T2, Y, 15, 13>
9 : <T2, commit>
10: <T3, start>
11: <T3, X, 7, 9>
```



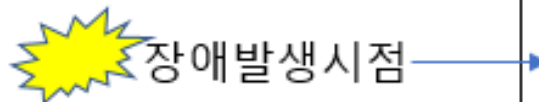
2. 장애와 회복

작업 요약

트랜잭션	즉시 갱신시	지연갱신시
T_1	조치불필요	조치불필요
T_2	redo	redo
T_3	undo	조치불필요

로그

```
1 : <T1, start>
2 : <T1, X, 0, 5>
3 : <T1, Y, 20, 15>
4 : <T1, commit>
5 : <checkpoint>
6 : <T2, start>
7 : <T2, X, 5, 7>
8 : <T2, Y, 15, 13>
9 : <T2, commit>
10: <T3, start>
11: <T3, X, 7, 9>
```



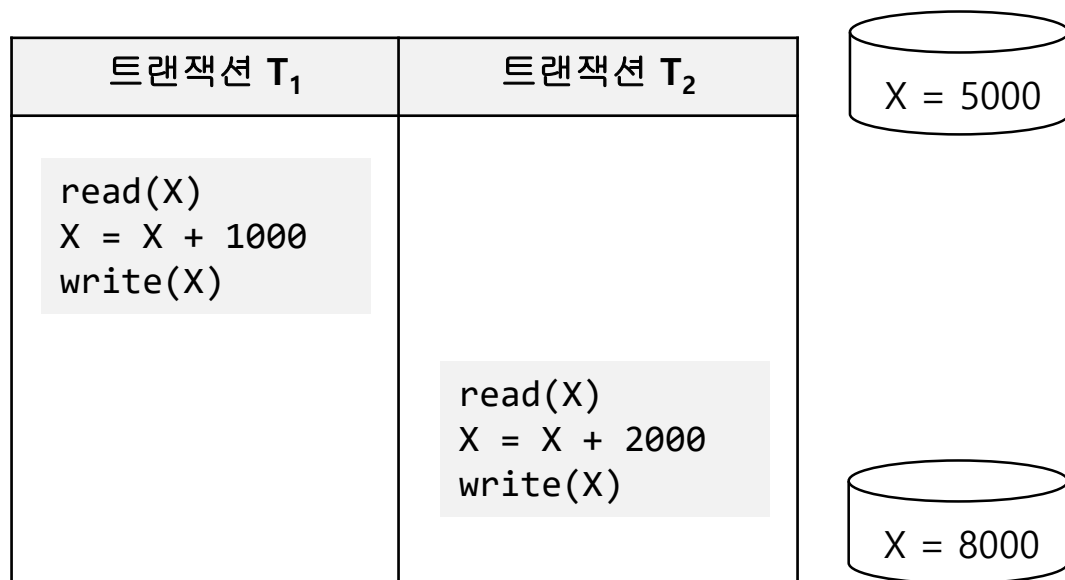
3. 병행 수행 제어

- 개요

- DBMS 는 다수 사용자의 요구를 빠르게 처리할수 있어야함
- 이를 위해 DBMS는 요청되는 트랜잭션을 한번에 하나씩 순차적으로 실행하는 것이 아니라 여러 트랜잭션들을 병행하여 수행
- 트랜잭션들을 병행 수행하면 주어진 시간에 많은 트랜잭션을 처리할 수 있게 되어 처리율(throughput)은 높아지지만 데이터베이스의 일관성에 문제가 생길 수 있다.
- DBMS는 트랜잭션의 4가지 특성을 유지하면서도 병행 수행을 해야 하는데, 이를 **병행 수행 제어(concurrency control)**라고 한다.
- 병행 수행 제어는 DBMS의 기능중 하나로서 병행 수행중인 트랜잭션들이 같은 데이터에 접근하여 연산을 실행하여도 문제가 발생하지 않고, 각각의 트랜잭션을 순차적으로 실행한 것과 같은 결과를 얻을 수 있도록 트랜잭션의 수행을 제어하는 것을 의미

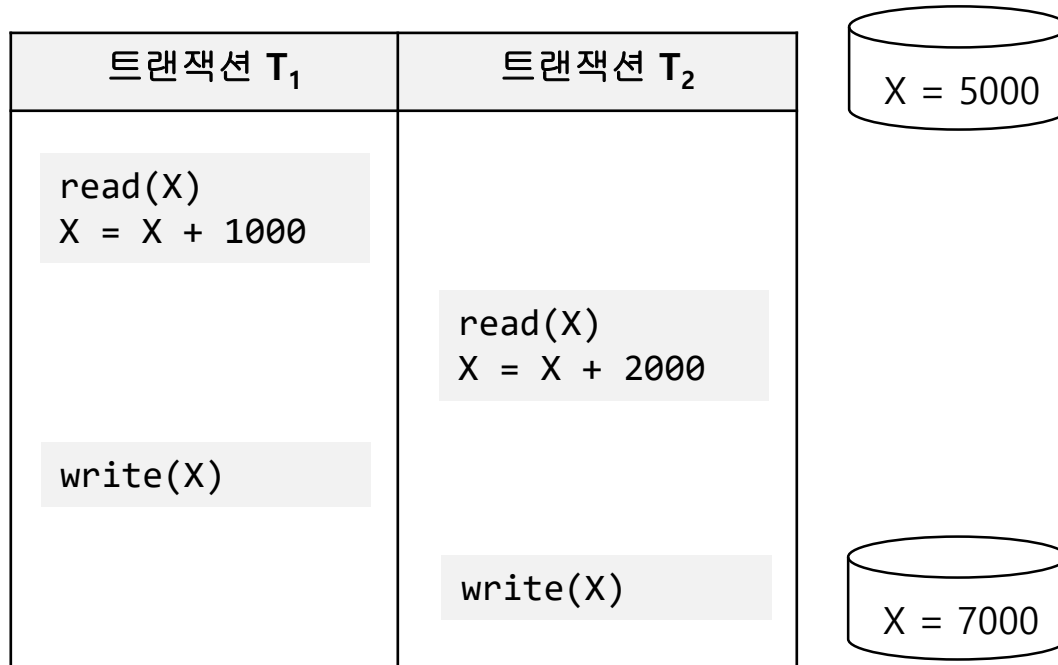
3. 병행 수행 제어

- 병행 수행시 발생할 수 있는 문제
 - 갱신분실, 모순성, 연쇄복귀의 문제가 발생 가능
 - 갱신분실 (lost update)
 - 하나의 트랜잭션이 수행한 데이터 변경 연산의 결과를 다른 트랜잭션이 재변경함으로써 이전 변경 연산이 무효화되는 현상



<그림 12-9> 트랜잭션의 순차적 실행 결과

3. 병행 수행 제어



<그림 12-10> 병행 수행에 의한 갱신 분실의 발생

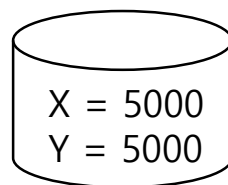
- 트랜잭션 T_2 의 실행 결과만 데이터베이스에 반영이 되고, 트랜잭션 T_1 의 실행은 무효화
- 트랜잭션 T_1 의 write(X) 연산의 결과가 분실

3. 병행 수행 제어

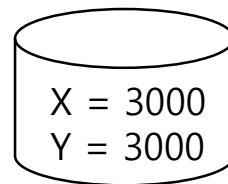
- 모순성 (inconsistency)

- 하나의 트랜잭션이 여러 개의 갱신 연산을 실행 할 때 서로 다른 상태의 데이터베이스를 참조하여 연산함으로써 데이터베이스에 모순된 결과를 초래하는 경우

트랜잭션 T_1	트랜잭션 T_2
<pre>read(X) X = X + 1000 write(X) read(Y) Y = Y + 1000 write(Y)</pre>	<pre>read(X) X = X * 0.5 write(X) read(Y) Y = Y * 0.5 write(Y)</pre>

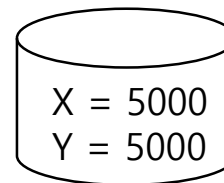


<그림 12-11> 트랜잭션의
순차적 실행 결과

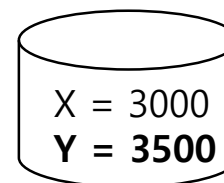


3. 병행 수행 제어

트랜잭션 T_1	트랜잭션 T_2
<pre>read(X) X = X + 1000 write(X)</pre>	<pre>read(X) X = X * 0.5 write(X) read(Y) Y = Y * 0.5 write(Y)</pre>
<pre>read(Y) Y = Y + 1000 write(Y)</pre>	

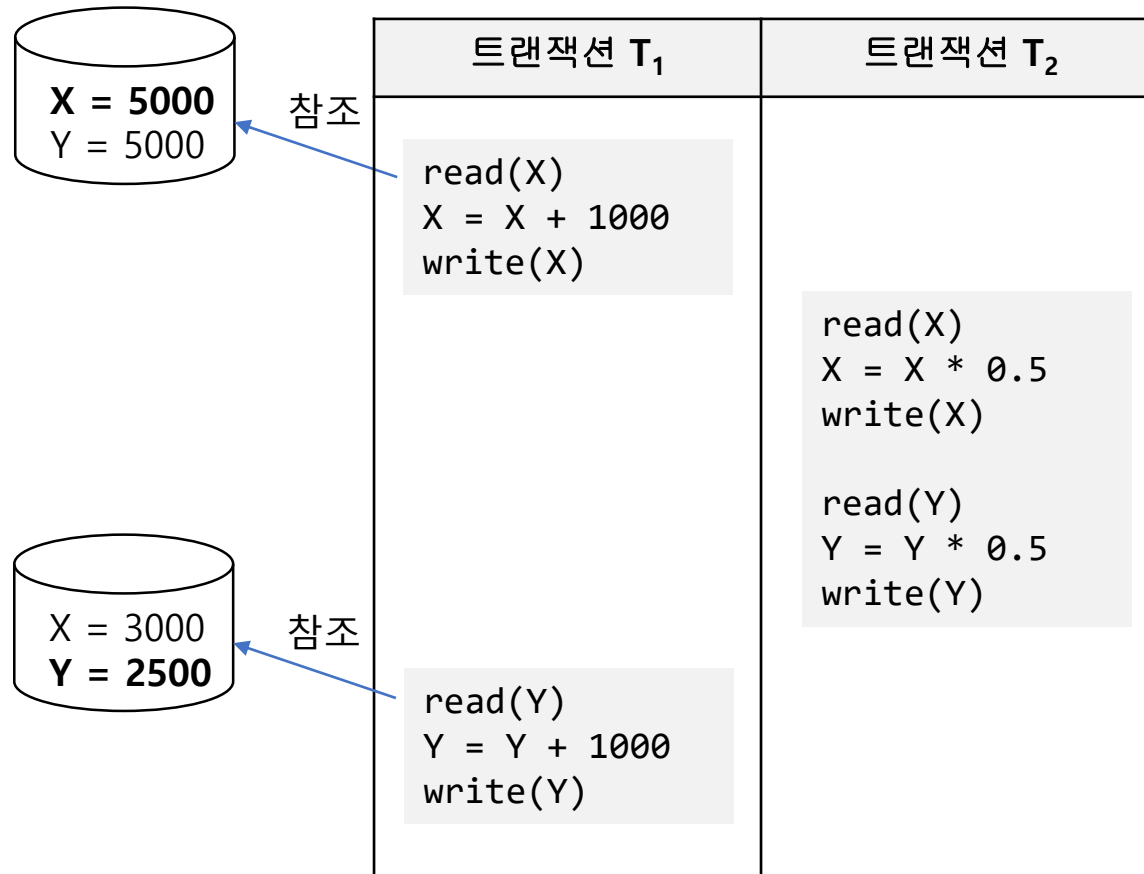


<그림 12-12> 병행 수행에 의한
모순성의 발생



트랜잭션 T_1 에서 읽어오는 X 와 Y 의 값이 동일시점의 값이어야 함에도 불구하고 X 는 트랜잭션이 수행되기 이전의 데이터베이스에서 읽어왔고, Y 는 트랜잭션 T_2 의 실행 결과가 반영된 데이터베이스에서 읽어왔기 때문에 두 값의 차이가 발생

3. 병행 수행 제어

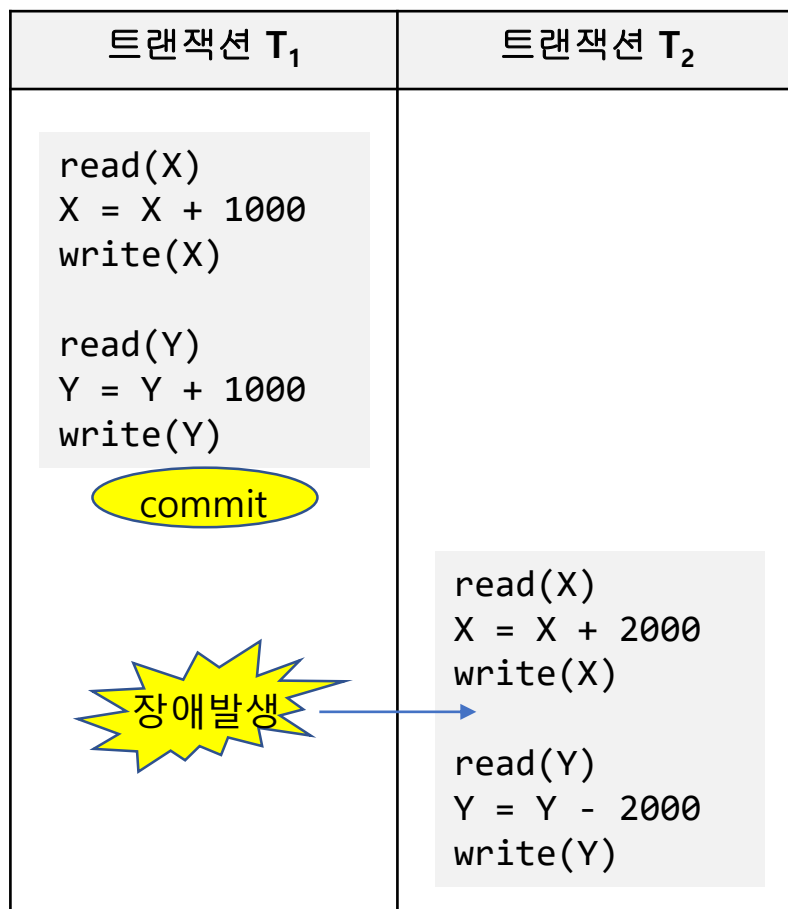


<그림 12-13> 서로 다른 상태의 데이터베이스를 참조하는 경우

3. 병행 수행 제어

연쇄복귀 (cascading rollback)

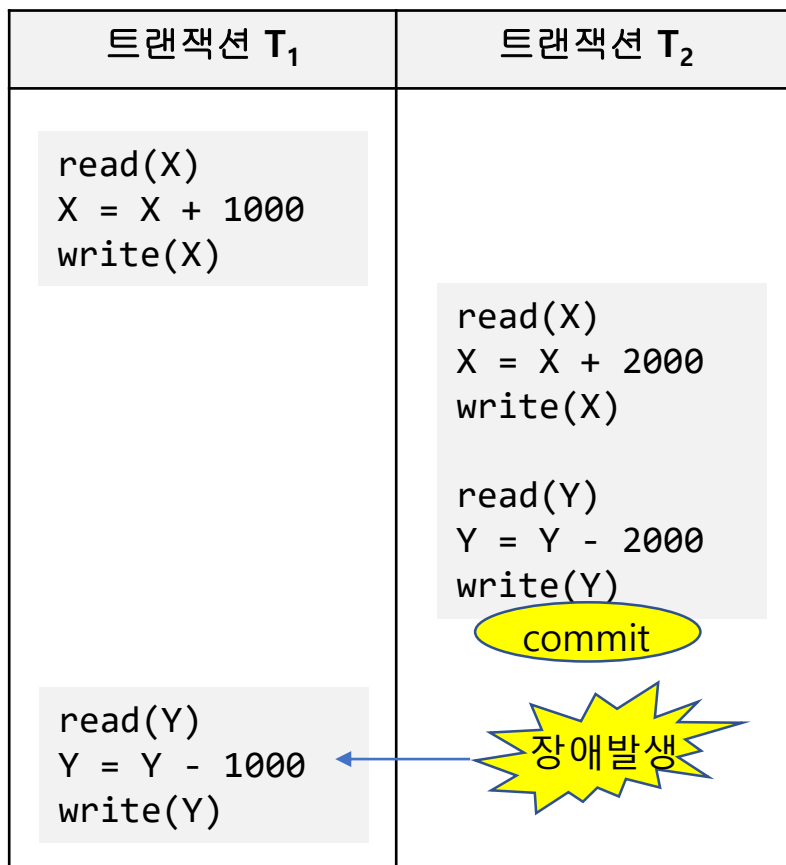
- 여러 트랜잭션이 상호 영향을 미치며 실행되다 보니 한 트랜잭션의 rollback은 연쇄적으로 다른 트랜잭션의 rollback을 유발하는 경우가 많음
- 이 때 연쇄적 rollback이 정상적으로 이루어지 못하는 상황을 연쇄복귀의 문제라고 한다



<그림 12-14> 트랜잭션의
순차적 실행시 장애 발생

트랜잭션 T_2 만 rollback
하면 문제 없음

3. 병행 수행 제어



<그림 12-15> 병행 수행에 의한
연쇄복귀 문제 발생

트랜잭션 T_1 이 rollback 됨에 따라, T_1 의 결과를 참조한 T_2 도 rollback 되어야 하나 정상완료(commit)가 되었기 때문에 트랜잭션의 지속성(durability) 특성에 따라 데이터베이스에 영구 반영되어야 하므로 rollback이 불가능

3. 병행 수행 제어

- 트랜잭션 스케줄

- 병행 수행시 트랜잭션들을 오고가면서 트랜잭션들을 구성하는 연산들을 번갈아 수행하는데 이런 방식을 **인터리빙(interlaving)**이라고 한다.
- 트랜잭션 스케줄(transaction schedule)** : 트랜잭션들을 구성하는 연산들을 어떤 순서로 실행할지에 대한 계획

<표 12-4> 트랜잭션 스케줄의 유형

스케줄의 종류	특징
직렬 스케줄	각 트랜잭션을 순차적으로 실행하는 스케줄 (인터리빙 X)
비직렬 스케줄	인터리빙 방식으로 트랜잭션들을 병행 수행하는 스케줄
직렬가능 스케줄	비직렬 스케줄 중에서 직렬 스케줄과 같이 올바른 결과를 도출하는 스케줄

Note. 트랜잭션 스케줄러(scheduler)

DBMS는 하나의 단일 SW이기보다는 다양한 기능을 수행하는 모듈들의 집합체라고 할 수 있다. 이 중에는 트랜잭션의 스케줄 작성을 담당하는 트랜잭션 스케줄러(scheduler)가 존재한다.

3. 병행 수행 제어

직렬 스케줄(serial schedule)

- 트랜잭션들을 하나하나 순차적으로 실행하는 방식
- 하나의 트랜잭션이 완전히 종료된 후에 다음 트랜잭션을 수행하기 때문에 트랜잭션들이 서로 영향을 미치는 일은 없다.
- 가장 단순한 스케줄링 기법으로 트랜잭션들의 수행 순서만 정해주면 된다.

트랜잭션 T_1	트랜잭션 T_2
<pre>read(X) X = X + 1000 write(X) read(Y) Y = Y + 1000 write(Y)</pre>	<pre>read(X) X = X + 2000 write(X) read(Y) Y = Y - 2000 write(Y)</pre>

<그림 12-16> 직렬 스케줄의 예

3. 병행 수행 제어

○ 비직렬 스케줄(nonserial schedule)

- 인터리빙 방식에 의해 트랜잭션들을 병행 수행하는 모든 종류의 수행계획
- 하나의 트랜잭션이 완료되기 전에 다른 트랜잭션이 시작되어 병행 수행 되며, 병행 수행중인 트랜잭션들이 동일 데이터를 참조하거나 갱신하게 되면 문제가 발생할 수 있다.

트랜잭션 T_1	트랜잭션 T_2
<pre>read(X) X = X + 1000 write(X)</pre>	<pre>read(X) X = X + 2000 write(X)</pre> <pre>read(Y) Y = Y - 2000 write(Y)</pre>
<pre>read(Y) Y = Y - 1000 write(Y)</pre>	

<그림 12-17> 비직렬
스케줄의 예

연쇄 복귀의 문제를 유발

3. 병행 수행 제어

- 직렬가능 스케줄(serializable schedule)

- 인터리빙 방식에 의해 트랜잭션들을 병행 수행하지만 병행 수행의 문제를 일으키지 않는 스케줄

트랜잭션 T_1	트랜잭션 T_2
<pre>read(X) X = X + 1000 write(X)</pre>	<pre>read(X) X = X + 2000 write(X)</pre>
<pre>read(Y) Y = Y - 1000 write(Y)</pre>	<pre>read(Y) Y = Y - 2000 write(Y)</pre>

<그림 12-18> 직렬가능
스케줄의 예

직렬 스케줄과 동일한 결과 도출

3. 병행 수행 제어

<표 12-5> 트랜잭션 스케줄의 유형 비교

스케줄의 종류	병행 수행 여부	트랜잭션 처리율	병행 수행시 문제 발생
직렬 스케줄	×	낮음	×
비직렬 스케줄	○	높음	△
직렬가능 스케줄	○	중간	×

- 비직렬 스케줄이 직렬 스케줄과 비슷해질수록 문제 발생 가능성은 줄어드는 대신 트랜잭션 처리율도 낮아진다.
- 반대로 트랜잭션들의 인터리빙 수행의 정도가 높아질수록 트랜잭션의 처리율을 높아지나 병행 수행 문제가 발생할 가능성도 높아진다.

3. 병행 수행 제어

- 병행 수행 제어 기법: 기본 로킹 규약
 - 병행 수행 제어의 목표는 결국 직렬가능 스케줄을 만드는 것
 - 많은 연구를 거쳐 직렬가능성을 보장하는 규약(protocol)이 개발
 - 로킹(locking) 규약이 대표적
 - 로킹 기법의 기본 원리는 하나의 트랜잭션이 먼저 접근한 데이터에 필요한 연산을 마칠 때 까지 **잠금(lock)**을 해 둠으로 해서 다른 트랜잭션의 접근을 막고, 연산을 마치면 잠금을 **해제(unlock)**하여 다른 트랜잭션이 사용할 수 있도록 하는 것
 - **잠금(lock)** 연산은 트랜잭션 T_1 이 데이터 X 에 대한 독점 사용권을 획득 할 때 사용
 - **해제(unlock)** 연산은 트랜잭션 T_1 이 데이터 X 에 대한 독점 사용권을 반납 할 때 사용

3. 병행 수행 제어

- 기본 로킹 규약

- ① 트랜잭션이 read하거나 write할 필요가 있는 데이터가 있다면 해당 데이터에 잠금 요청을 하여 독점 사용권을 획득한 후에 데이터에 접근할 수 있다.
- ② 트랜잭션이 필요한 연산을 마치면 잠금을 해제한다.
- ③ 잠금이 된 데이터는 독점 사용권을 가진 트랜잭션 외에는 접근이 불가능하다.

3. 병행 수행 제어

트랜잭션 T ₁	트랜잭션 T ₂
<pre>lock(X) read(X) X = X + 1000 write(X) unlock(X)</pre>	<pre>lock(X) read(X) X = X + 2000 write(X) unlock(X)</pre>
<pre>lock(Y) read(Y) Y = Y - 1000 write(Y) unlock(Y)</pre>	<pre>lock(Y) read(Y) Y = Y - 2000 write(Y) unlock(Y)</pre>

<그림 12-19> 기본
로킹 기법을 이용한
스케줄의 예

3. 병행 수행 제어

로킹(locking)의 단위

- 로킹의 대상이 되는 객체의 크기를 로킹 단위라고 한다
- 데이터베이스 전체, 테이블, 테이블의 튜플이나 컬럼까지도 가능
- 로킹의 단위가 커질수록 트랜잭션의 제어는 쉽지만 병행성은 낮아진다.
- 반면 로킹의 단위가 작아지면 병행성은 높아지지만 제어가 어렵게 된다.

<표 12-6> 로킹 단위에 따른 장단점 비교

로킹단위	병행 제어	병행 수행 수준	로킹 오버헤드	DB 공유정도
커짐	단순해짐	낮아짐	감소	낮아짐
작아짐	복잡해짐	높아짐	증가	높아짐

3. 병행 수행 제어

● 기본 로킹(locking) 규약의 문제점

- 로킹에 의한 데이터 독점 사용권의 보장이라는 정책이 지나치게 엄격하여 병행 수행성을 저해한다
- 기본 로킹 규약만으로는 작성된 스케줄의 직렬 가능성을 완벽히 보장하지 못한다

<병행수행성 높이기>

- 단순히 데이터 X를 읽기(read)만 하는 두 트랜잭션 T_1 과 T_2 가 있다고 하면 두 트랜잭션은 데이터베이스에 아무 변화도 일으키지 않기 때문에 동시에 수행이 된다고 해도 아무 문제가 없다.
 - 따라서 read 연산에 대한 규제를 완화하면 병행 수행의 수준을 높일 수 있다
 - read 연산을 위한 로크(lock)와 write 연산을 위한 로크(lock)를 구분하여 사용하는 방안이 제시

3. 병행 수행 제어

<표 12-7> 공용 로크와 전용 로크

로크의 종류	설명
공용 로크 (shared lock)	<ul style="list-style-type: none">·트랜잭션 T_1이 데이터 X에 공용 로크를 설정하면 T_1은 데이터 X에 대해 read 연산만 가능하다. (write 연산은 불가)·다른 트랜잭션들도 동시에 데이터 X에 공용 로크를 설정하는 것이 가능하다.·결국 데이터 X에 대한 읽기 권한을 여러 트랜잭션이 함께 가질 수 있다
전용 로크 (exclusive lock)	<ul style="list-style-type: none">·트랜잭션 T_1이 데이터 X에 전용 로크를 설정하면 T_1은 데이터 X에 대해 read, write 연산이 모두 가능하다.·전용 로크는 T_1에 독점 사용권이 주어지므로 다른 트랜잭션들은 데이터 X에 대해 공용 로크나 전용 로크를 설정할 수 없다.

3. 병행 수행 제어

<표 12-8> 공용 로크와 전용 로크

		다른 트랜잭션들	
		공용 로크	전용 로크
트랜잭션 T_1	공용 로크	가능	불가능
	전용 로크	불가능	불가능

Note. 공용/전용 로크 선택에 대한 판단

로크는 트랜잭션 스케줄러에 의해 트랜잭션의 실행 스케줄이 작성될 때 스케줄러가 설정하는 것이다. 따라서 공용 로크로 설정할지 전용 로크로 설정할지의 여부도 스케줄러가 판단하여 결정한다.

3. 병행 수행 제어

- 기본 로킹 규약이 직렬성을 보장하지 못하는 예

초기값 : $X=5000$, $Y=5000$

직렬 스케줄(T_1, T_2) 실행 결과
 $X=3000$, $Y=3000$

<그림 12-20> 스케줄 실행 결과
 $X=3000$, $Y=3500$

트랜잭션 T_1	트랜잭션 T_2
<pre>lock(X) read(X) X = X + 1000 write(X) unlock(X)</pre>	<pre>lock(X) read(X) X = X * 0.5 write(X) unlock(X) lock(Y) read(Y) Y = Y * 0.5 write(Y) unlock(Y)</pre>
<pre>lock(Y) read(Y) Y = Y + 1000 write(Y) unlock(Y)</pre>	

<그림 12-20>

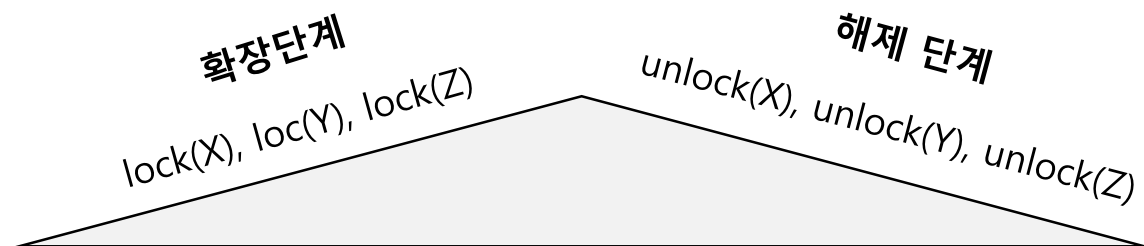
3. 병행 수행 제어

- 병행 수행 제어 기법: 2단계 로킹 규약

- 2단계 로킹 규약(2 phase locking protocol)은 기본 로킹 규약의 한계를 극복하기 위해서 잠금(lock)과 해제(unlock)의 수행 시점에 대한 규칙을 추가
- 잠금(lock)과 해제(unlock)를 다음과 같이 2단계에 맞추어 수행

확장단계	트랜잭션이 잠금(lock) 연산을 수행하는 단계 (해제(unlock)의 수행은 불가)
축소단계	트랜잭션이 해제(unlock) 연산을 수행하는 단계 (잠금(lock)의 수행은 불가)

- 간단히 요약하면 잠금(lock)을 할 때는 계속 잠금(lock)만 하고, 해제(unlock)를 할 때는 계속 해제(unlock)만 해야 한다는 의미
- 모든 트랜잭션의 실행이 2단계 로킹 규약을 따르면 항상 직렬 가능 스케줄을 얻을 수 있다고 알려져 있다.



3. 병행 수행 제어

트랜잭션 T_1	트랜잭션 T_2
<pre> lock(X) read(X) X = X + 1000 write(X) lock(Y) unlock(X) </pre>	<pre> lock(X) read(X) X = X * 0.5 write(X) </pre>
<pre> read(Y) Y = Y + 1000 write(Y) unlock(Y) </pre>	<pre> lock(Y) unlock(X) read(Y) Y = Y * 0.5 write(Y) unlock(Y) </pre>

초기값 : $X=5000, Y=5000$

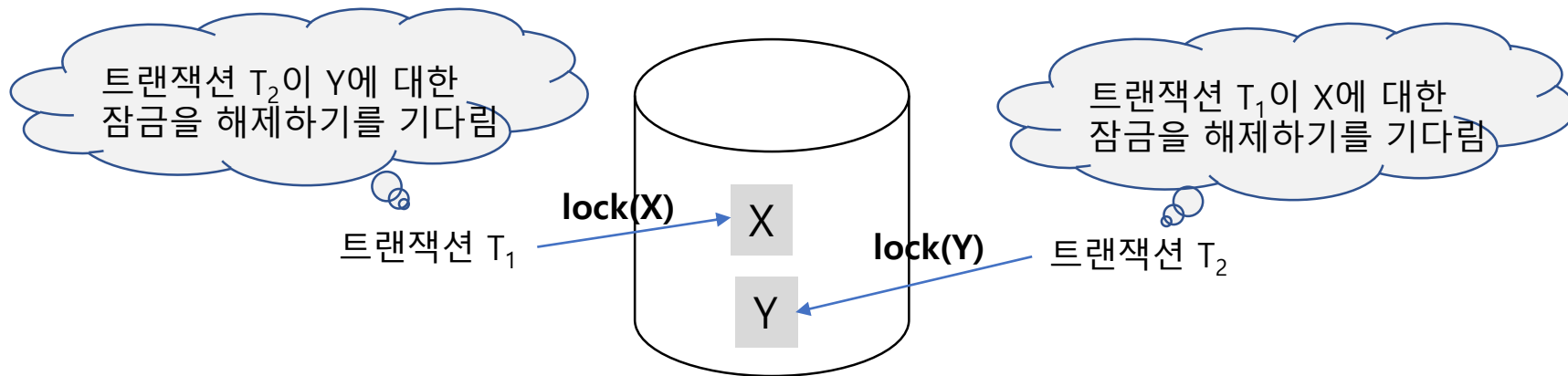
직렬 스케줄(T_1, T_2) 실행 결과
 $X=3000, Y=3000$

<그림 12-22> 스케줄 실행 결과
 $X=3000, Y=3000$

<그림 12-22> 2단계 로킹 규약에
따른 트랜잭션 스케줄의 예

3. 병행 수행 제어

- 교착 상태(dead lock)



<그림 12-23> 교착 상태의 예

- 교착 상태를 해결하는 방법

- 예방기법 : 아예 교착 상태가 발생하지 않도록 하는 방법
- 회피 기법 : 교착 상태가 발생하지 않도록 스케줄을 조정하는 방법 (ex. 트랜잭션 수행할 때 타임스탬프를 설정하고 상대방 트랜잭션이 로킹을 해제하기를 기다리는 시간이 제한 시간을 넘어가면 rollback 하여 트랜잭션을 철회하고 스케줄에 새로 진입)

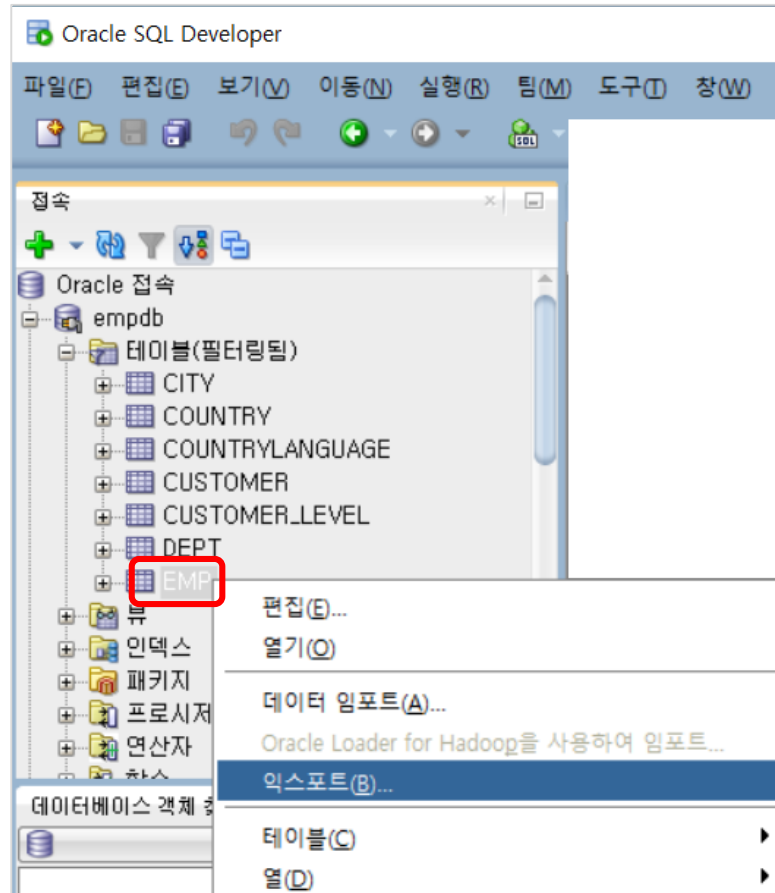
실습. 테이블을 엑셀 파일로 저장

- 실습 개요
 - 테이블의 데이터를 엑셀 파일로 저장하기
 - 엑셀 파일의 데이터를 테이블에 저장하기

실습. 테이블을 엑셀 파일로 저장

- 테이블 데이터를 엑셀파일로 저장하기

- emppdb 데이터베이스의 emp 테이블 데이터를 엑셀 파일로 저장해 보자
- ① 접속창에서 저장 대상 테이블을 선택한 뒤 마우스 오른쪽 버튼을 클릭하여 팝업메뉴가 표시되면 [엑스포트]를 선택한다.



실습. 테이블을 엑셀 파일로 저장

- ② 익스포트 마법사가 표시되면 다음과 같이 선택 또는 입력한 후 [다음] 버튼을 클릭한다.

항목	선택 또는 입력 값
DDL 익스포트	선택을 해제한다. (DDL 익스포트는 데이터를 SQL문 형태로 내보낼 때 사용)
형식	excel 2003* (xlsx)
질의 워크시트	선택을 해제한다.
파일	테이블 데이터를 저장할 폴더와 파일 이름을 입력한다

익스포트 마법사 - 단계 1/3

소스/대상

소스/대상

데이터 지정

익스포트 요약

접속(C): empdb

☐ DDL 익스포트(E)

☒ 데이터 익스포트(O)

형식(F): excel 2003* (xlsx) ☒ 헤더

데이터 워크시트 이름(N):

☐ 질의 워크시트 이름(Q):

다른 이름으로 저... 단일 파일 ☐ 압축됨(B) 인코딩(I): MS949

파일(F): D:\TestWemp.xlsx

☐ 요약으로 이동합니다(M).

다음(N) >

실습. 테이블을 엑셀 파일로 저장

- ③데이터 선택 조건이 있으면 입력하고 [다음] 버튼을 클릭한다. (여기서는 모든 데이터를 저장할 것이므로 아무 조건도 입력하지 않았다)

엑스포트 마법사 - 단계 2/3

데이터 지정

소스/대상
데이터 지정
엑스포트 요약

데이터베이스 객체	열	객체 Where
SCOTT.EMP	*	

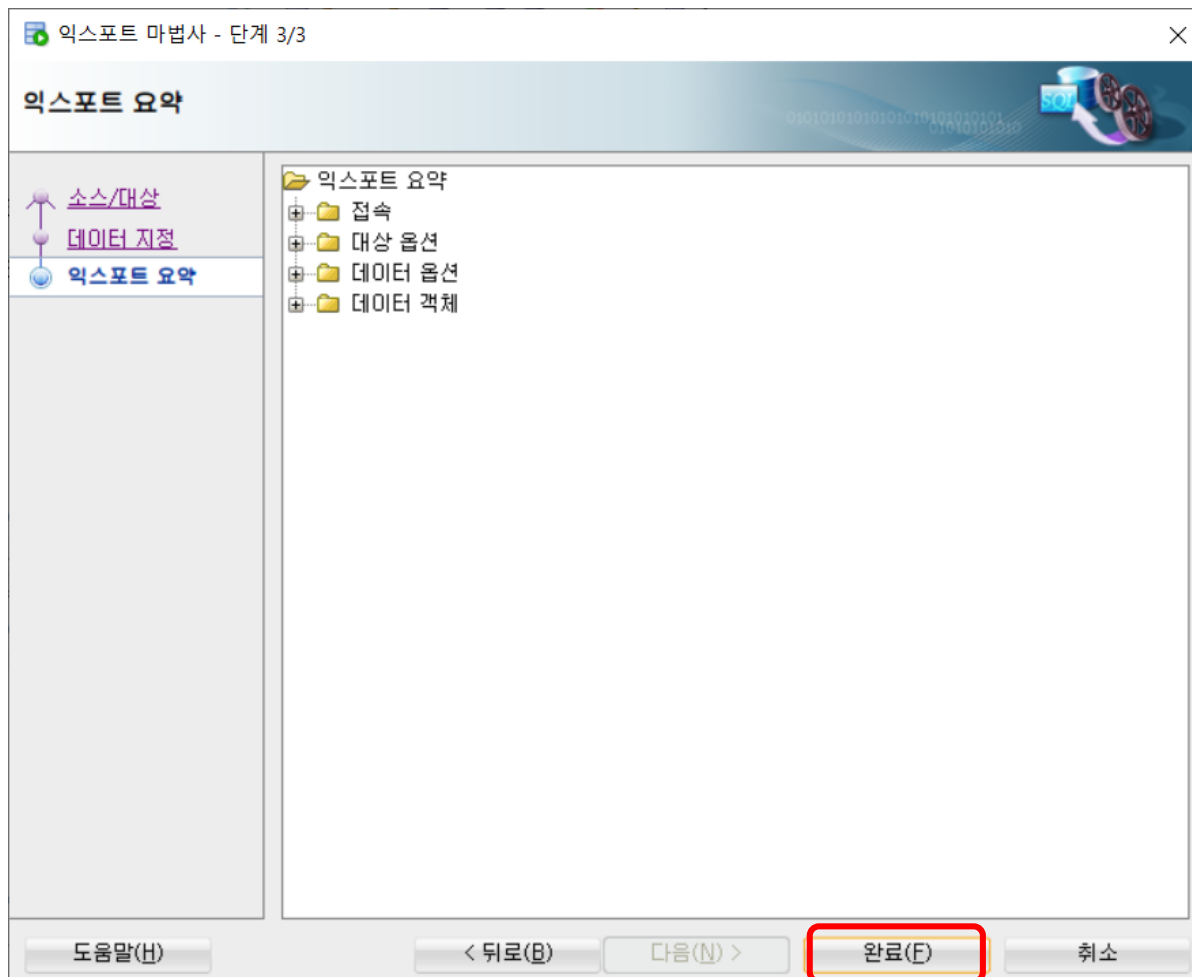
데이터를 선택해서 저장하려면 여기에 선택 조건을 서술한다.
(예) deptno = 10

전역 Where:

도움말(H) < 뒤로(B) 다음(N) > 완료(F) 취소

실습. 테이블을 엑셀 파일로 저장

- ④ [완료] 버튼을 클릭하여 엑셀파일 저장 작업을 실행한다. (저장할 데이터의 양에 따라 약간의 시간이 소요된다)



실습. 테이블을 엑셀 파일로 저장

- ⑤ 엑셀 파일을 열어서 데이터가 저장된 것을 확인한다.

	A	B	C	D	E	F	G	H	
1	EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO	
2	7839	KING	PRESIDENT		11-17-1981	5000		10	
3	7698	BLAKE	MANAGER	7839	5-1-1981	2850		30	
4	7782	CLARK	MANAGER	7839	6-9-1981	2450		10	
5	7566	JONES	MANAGER	7839	4-2-1981	2975		20	
6	7654	MARTIN	SALESMAN	7698	8-28-1981	1250	1400	30	
7	7499	ALLEN	SALESMAN	7698	2-20-1981	1600	300	30	
8	7844	TURNER	SALESMAN	7698	8-8-1981	1500	0	30	
9	7900	JAMES	CLERK	7698	12-3-1981	950		30	
10	7521	WARD	SALESMAN	7698	2-22-1981	1250	500	30	
11	7902	FORD	ANALYST	7566	12-3-1981	3000		20	

저장 파일의 형식

테이블 데이터를 저장할 때 다양한 포맷의 파일로 저장이 가능하다. CSV, HTML, TEXT 등 다양한 형식으로 저장할 수 있다.

실습. 테이블을 엑셀 파일로 저장

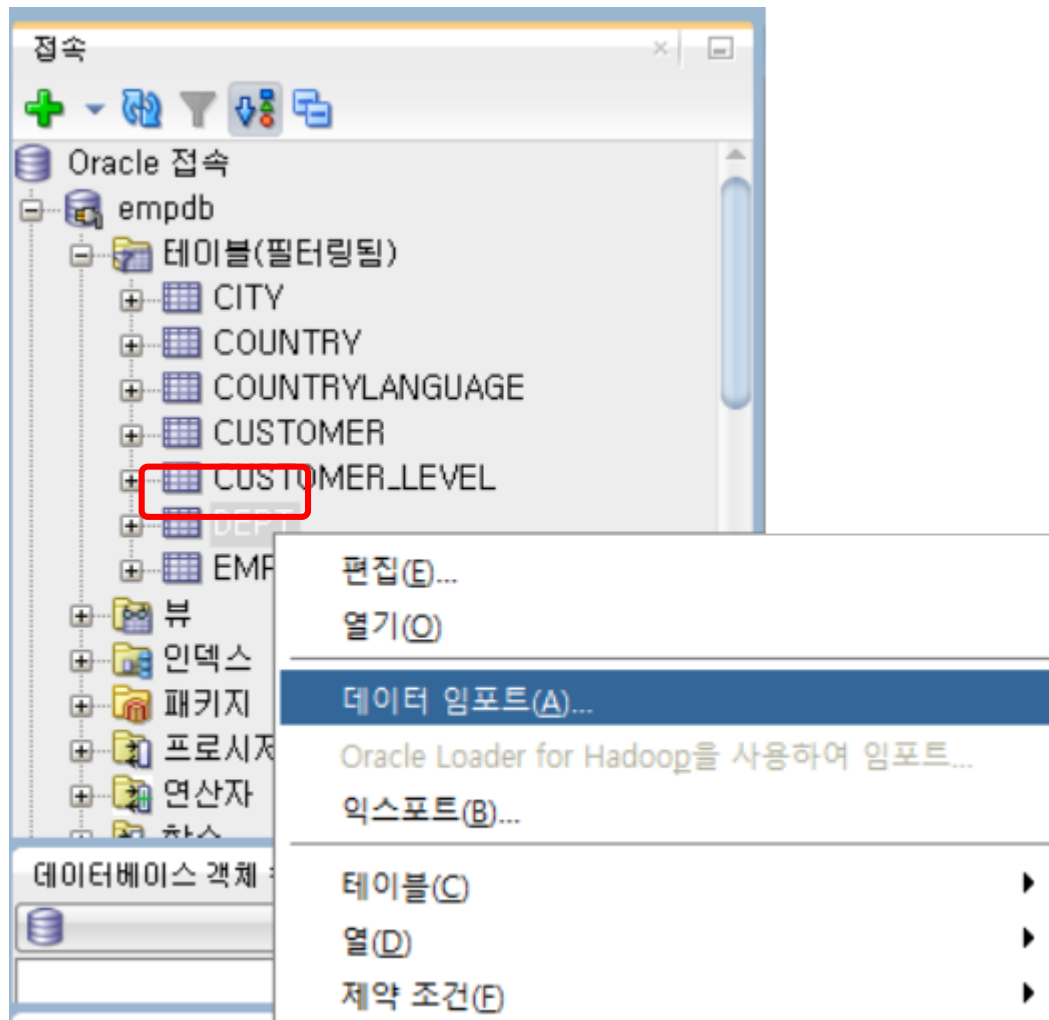
- 엑셀파일 데이터 테이블에 저장하기

- 부서정보 데이터를 엑셀파일로 작성한 뒤 이를 읽어서 DEPT 테이블에 저장해 보자.
- ① 엑셀 파일에 다음과 같이 입력하여 dept.xlsx 에 저장하자.

	A	B	C	D
1	부서번호	부서명	부서위치	
2	91	HR 1 TEAM	SEOUL	
3	92	HR 2 TEAM	BUSAN	
4	93	HR 3 TEAM	INCHON	
5	94	HR 4 TEAM	DAEGOO	
6				

실습. 테이블을 엑셀 파일로 저장

- ② dept 테이블의 팝업 메뉴에서 [데이터 импорт]를 선택한다.



실습. 테이블을 엑셀 파일로 저장

- ③ 데이터 импорт 마법사가 표시되면 읽어올 파일을 지정하고 '헤더' 항목을 체크 후 [다음] 버튼을 클릭한다. '헤더'는 엑셀 파일에서 첫줄이 데이터가 아닌 항목을 설명하는 이름 (컬럼이름)을 나타낸다 따라서 헤더 없이 바로 첫줄부터 데이터가 시작되는 경우는 '헤더' 항목의 체크를 해제 해야 한다.

데이터 импорт 마법사 - 단계 1/5

데이터 미리보기

데이터 미리보기

소스: 로컬 파일

파일: D:\test\dept.xlsx

파일 형식

☒ 헤더

행 건너뛰기(K): 0

형식(F): excel 95-2003 (*.xls)

☒ 미리보기 행 제한(P): 100

파일 내용

부서번호	부서명	부서위치
91	HR 1 TEAM	SEOUL
92	HR 2 TEAM	BUSAN
93	HR 3 TEAM	INCHON
94	HR 4 TEAM	DAEGOO

도움말(H) < 뒤로(B) 다음(N) > 완료(F) 취소

실습. 테이블을 엑셀 파일로 저장

- ④ 임포트 방식을 '삽입' 으로 선택한 뒤 [다음] 버튼을 클릭한다.

데이터 임포트 마법사 - 단계 2/4

임포트 방식

데이터 임포트 방식을 지정하십시오. 스테이징 외부 테이블 방식의 경우 대상 테이블을 임포트할 수 있도록 외부 테이블이 스테이징 테이블로 생성됩니다. 기타 임포트 방식의 경우 데이터가 테이블로 바로 임포트됩니다.

임포트 방식: **삽입**

☐ SQL 워크시트로 생성 스크립트 전송

테이블 이름:

☐ 임포트 행 제한(I):

파일 내용

부서번호	부서명	부서위치
91	HR 1 TEAM	SEOUL
92	HR 2 TEAM	BUSAN
93	HR 3 TEAM	INCHON
94	HR 4 TEAM	DAEGOO

도움말(H) < 뒤로(B) **다음(N) >** 완료(F) 취소

실습. 테이블을 엑셀 파일로 저장

- ⑤ 저장할 열을 선택하는 화면이다. 이미 필요한 열이 모두 선택되어 있으므로 [다음] 버튼을 클릭한다. 이 화면에서 열의 순서도 조정할 수 있다. 테이블의 열의 순서에 맞추는 것이 편리하다.

데이터 임포트 마법사 - 단계 3/5

열 선택

데이터 집합에서 임포트할 열을 선택하여 원하는 순서대로 배열하십시오.
사용 가능한 열

선택된 열
부서번호
부서명
부서위치

파일 내용

부서번호	부서명	부서위치
91	HR 1 TEAM	SEOUL
92	HR 2 TEAM	BUSAN
93	HR 3 TFAM	INCHON

도움말(H) < 뒤로(B) **다음(N) >** 완료(F) 취소

실습. 테이블을 엑셀 파일로 저장

- ⑥ 엑셀 파일의 열과 테이블의 열을 연결하는 과정이다. 모든 열별로 하나하나 연결을 해주어야 한다. (부서번호→DEPTNO, 부서명→DNAME, 부서위치→LOC)

데이터 임포트 마법사 - 단계 4/5

열 정의

왼쪽의 소스 데이터 열 목록에 있는 각 열에 대해 오른쪽의 대상 테이블을 선택합니다.

일치 기준 이름 ▼

소스 데이터 열

부서번호
부서명
부서위치

엑셀 파일의 열

대상 테이블 열

이름 DEPTNO ▼

데이터 유형 NUMBER

크기/전체 자릿수 2

소수점 이하 자릿수 0

☐ 널 가능? 기본값

설명

데이터

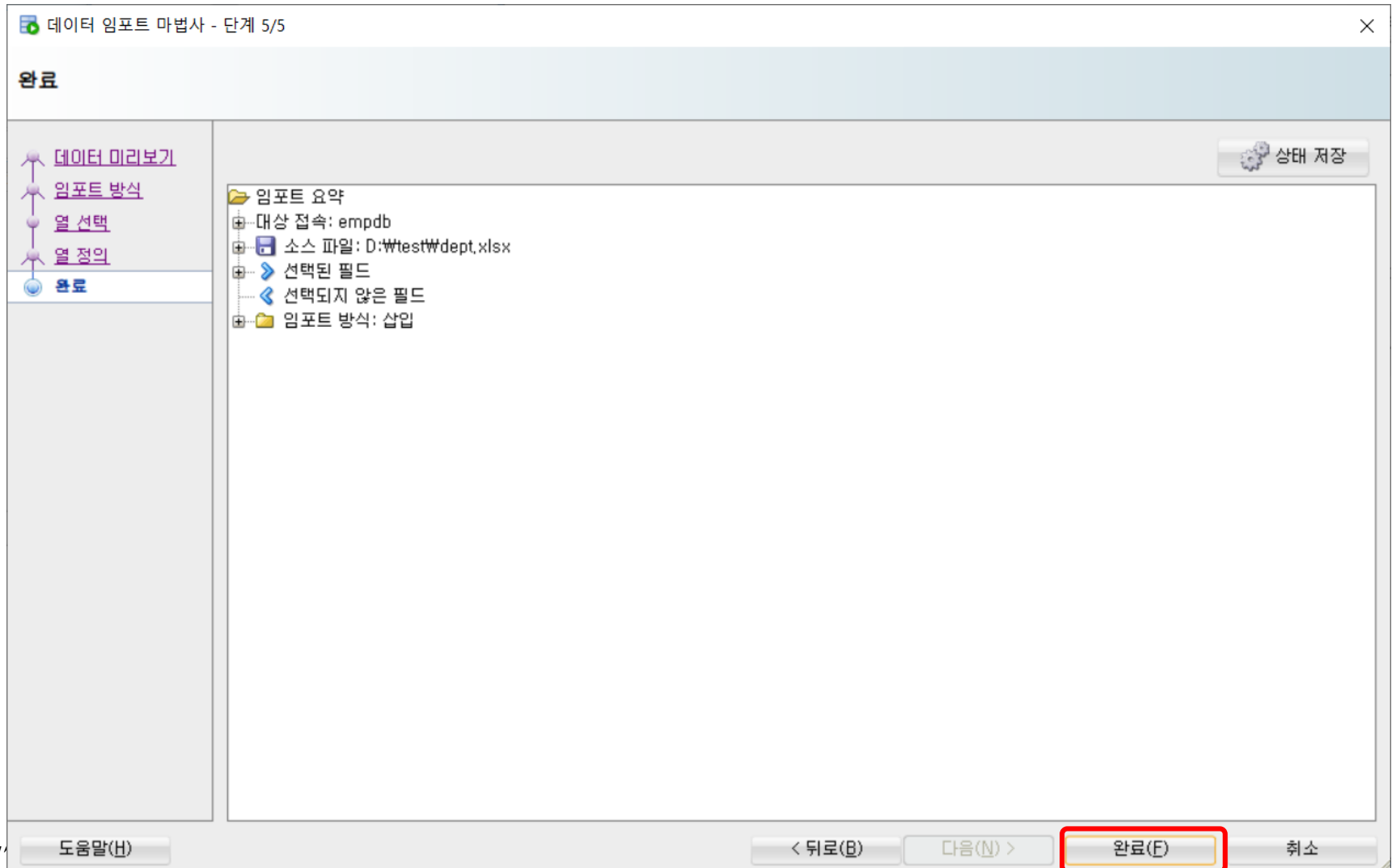
91
92
93
94

상태

도움말(H) < 뒤로(B) 다음(N) > 완료(F) 취소

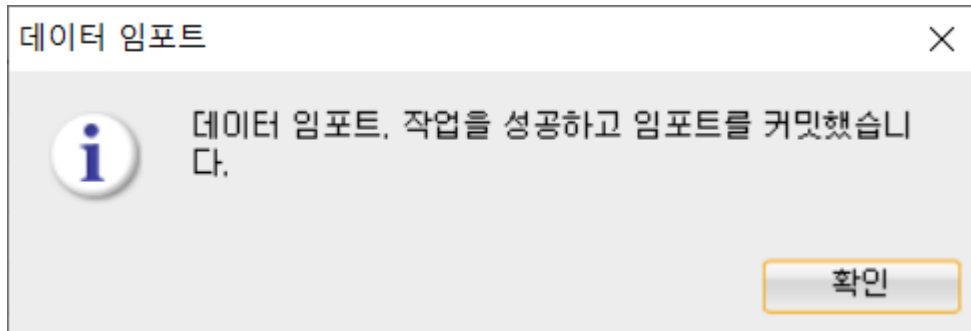
실습. 테이블을 엑셀 파일로 저장

- ⑦ [완료] 버튼을 클릭하여 импорт 작업을 실행한다.



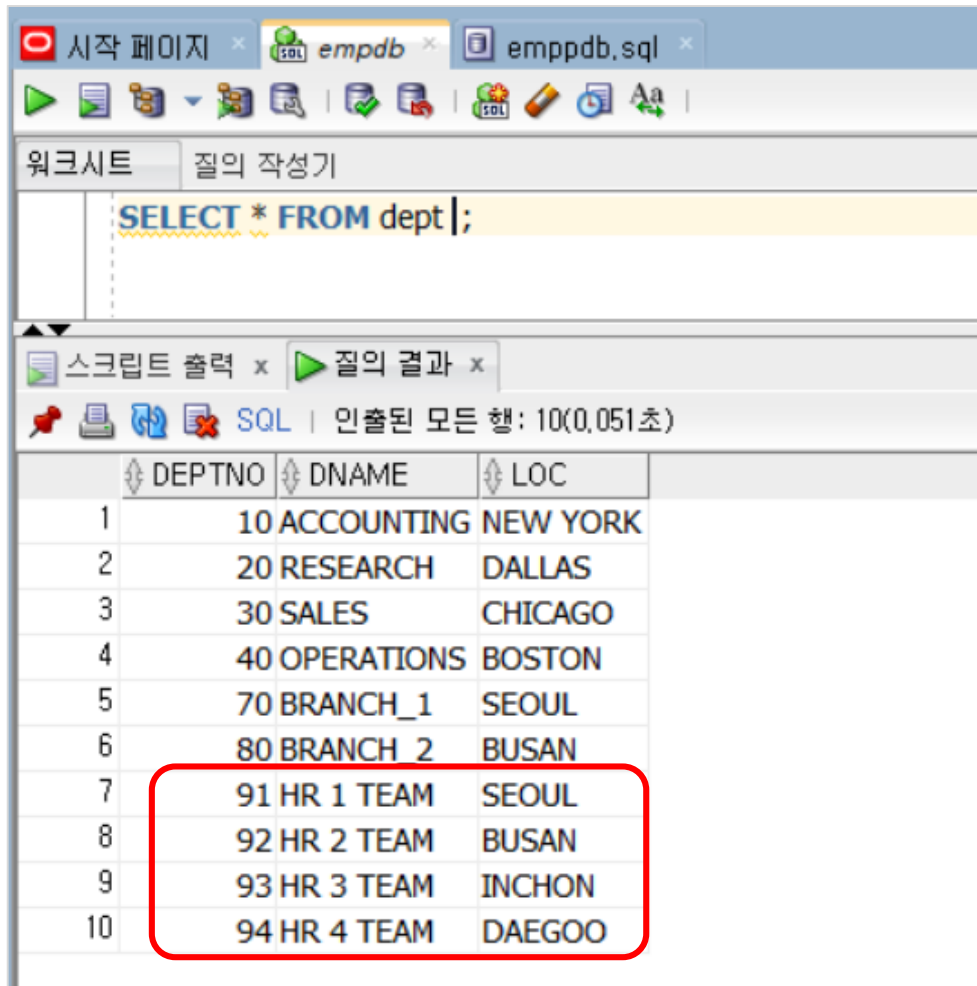
실습. 테이블을 엑셀 파일로 저장

- ⑨ 임포트 작업이 성공하면 다음의 메시지가 출력된다.



실습. 테이블을 엑셀 파일로 저장

- ⑨ SELECT 문을 통해 데이터가 테이블에 올바르게 저장되었는지 확인한다.



The screenshot shows a SQL IDE interface. The top toolbar includes icons for running queries, saving, and other database functions. The main window displays a SQL script with the query `SELECT * FROM dept;`. Below the script, the '질의 결과' (Query Results) tab is active, showing a table with 10 rows and 3 columns: DEPTNO, DNAME, and LOC. The last four rows of the table are highlighted with a red rectangle.

	DEPTNO	DNAME	LOC
1	10	ACCOUNTING	NEW YORK
2	20	RESEARCH	DALLAS
3	30	SALES	CHICAGO
4	40	OPERATIONS	BOSTON
5	70	BRANCH_1	SEOUL
6	80	BRANCH_2	BUSAN
7	91	HR 1 TEAM	SEOUL
8	92	HR 2 TEAM	BUSAN
9	93	HR 3 TEAM	INCHON
10	94	HR 4 TEAM	DAEGOO