



11장. 트랜잭션 관리

이번 장에서는 데이터베이스에서 가장 중요한 개념인 트랜잭션에 대한 개념을 이해하고, 오라클에서 트랜잭션을 구분하기 위해서 제공하는 명령어인 COMMIT과 ROLLBACK에 대해서 살펴보도록 하겠습니다.

이 장에서 다룰 내용



- 1 **트랜잭션**
.....●
- 2 **COMMIT과 ROLLBACK**
.....●
- 3 **자동 커밋**
.....●
- 4 **트랜잭션을 작게 분할하는 SAVEPOINT**
.....●

01. 트랜잭션



- ❖ 데이터베이스에서 트랜잭션(Transaction)은 데이터 처리의 한 단위입니다.
- ❖ 오라클에서 발생하는 여러 개의 SQL 명령문들을 하나의 논리적인 작업 단위로 처리하는데 이를 트랜잭션이라고 합니다.
- ❖ 하나의 트랜잭션은 All-OR-Nothing 방식으로 처리됩니다.
- ❖ 여러 개의 명령어의 집합이 정상적으로 처리되면 정상 종료하도록 하고 여러 개의 명령어 중에서 하나의 명령어라도 잘못되었다면 전체를 취소해버립니다.
- ❖ 데이터베이스에서 작업의 단위로 트랜잭션이란 개념을 도입한 이유는 데이터의 일관성을 유지하면서 안정적으로 데이터를 복구시키기 위해서입니다.

01. 트랜잭션



- ❖ 은행 현금인출기(ATM)에서 돈을 인출하는 과정으로 트랜잭션을 설명해 보기로 하겠습니다.

현금인출을 하겠다고 기계에게 알려준다.

현금카드를 넣어서 본인임을 인증 받는다.

인출할 금액을 선택하면 은행 현금인출기는 돈을 내어준다.

계좌에서 인출된 금액만큼을 잔액에서 차감한다.

01. 트랜잭션



- ❖ 이러한 거래에 있어서 지켜져야 할 중요한 것이 있습니다.
- ❖ 기계의 오동작 등으로 인하여 전산 상으로는 돈을 인출한 것으로 입력이 되었는데 돈은 안 나온다거나, 돈은 나왔는데 일련의 에러나 문제로 인하여서 돈을 인출한 것이 전산 상으로 입력이 안 되면 상당히 심각한 문제가 발생합니다.
- ❖ 이 때문에 전산 상으로도 입력이 정상적으로 잘 되고, 돈도 인출이 정상적으로 잘 됨을 확인하고 나서야, 인출하는 하나의 과정이 정상적으로 처리되었음을 확인할 수 있습니다.
- ❖ 여기서 돈을 인출하는 일련의 과정이 하나의 묶음으로 처리되어야 한다는 것을 이해할 수 있을 것입니다.
- ❖ 그리고 혹시 처리도중 중간에 무슨 문제가 발생한다면 진행되던 인출과정 전체를 취소하고 다시 처음부터 시작해야 합니다.
- ❖ 이것을 트랜잭션이라고 합니다.

01. 트랜잭션



- ❖ 트랜잭션 제어를 위한 명령어(Transaction Control Language)에는 다음과 같은 것들이 있습니다.

COMMIT
SAVEPOINT
ROLLBACK

02. COMMIT과 ROLLBACK



- ❖ 앞장에서 데이터를 추가, 수정, 삭제하는 작업들을 학습했는데, 이러한 데이터를 조작하는 명령어인 DML(Data Manipulation Language)은 이들이 실행됨과 동시에 트랜잭션이 진행됩니다.
- ❖ 이들 DML 작업이 성공적으로 처리되도록 하기 위해서는 COMMIT 명령을, 작업을 취소하기 위해서는 ROLLBACK 명령으로 종료해야 합니다.
- ❖ COMMIT은 모든 작업들을 정상적으로 처리하겠다고 확정하는 명령어로 트랜잭션의 처리 과정을 데이터베이스에 모두 반영하기 위해서 변경된 내용을 모두 영구 저장합니다.
- ❖ COMMIT 명령어를 수행하게 되면 하나의 트랜잭션 과정을 종료하게 됩니다.

02. COMMIT과 ROLLBACK



- ❖ ROLLBACK은 작업 중 문제가 발생되어서 트랜잭션의 처리 과정에서 발생한 변경사항을 취소하는 명령어입니다.
- ❖ ROLLBACK 명령어 역시 트랜잭션 과정을 종료하게 됩니다.
- ❖ ROLLBACK은 트랜잭션으로 인한 하나의 묶음 처리가 시작되기 이전의 상태로 되돌립니다.
- ❖ 트랜잭션은 여러 개의 물리적인 작업(DML 명령어)들이 모여서 이루어지는데 이러한 과정에서 하나의 물리적인 작업이라도 문제가 발생하게 되면 모든 작업을 취소해야 하므로 이들을 하나의 논리적인 작업 단위(트랜잭션)로 구성해 놓는다.
- ❖ 문제가 발생하게 되면 이 논리적인 작업 단위를 취소해 버리면 되기 때문입니다.

02. COMMIT과 ROLLBACK

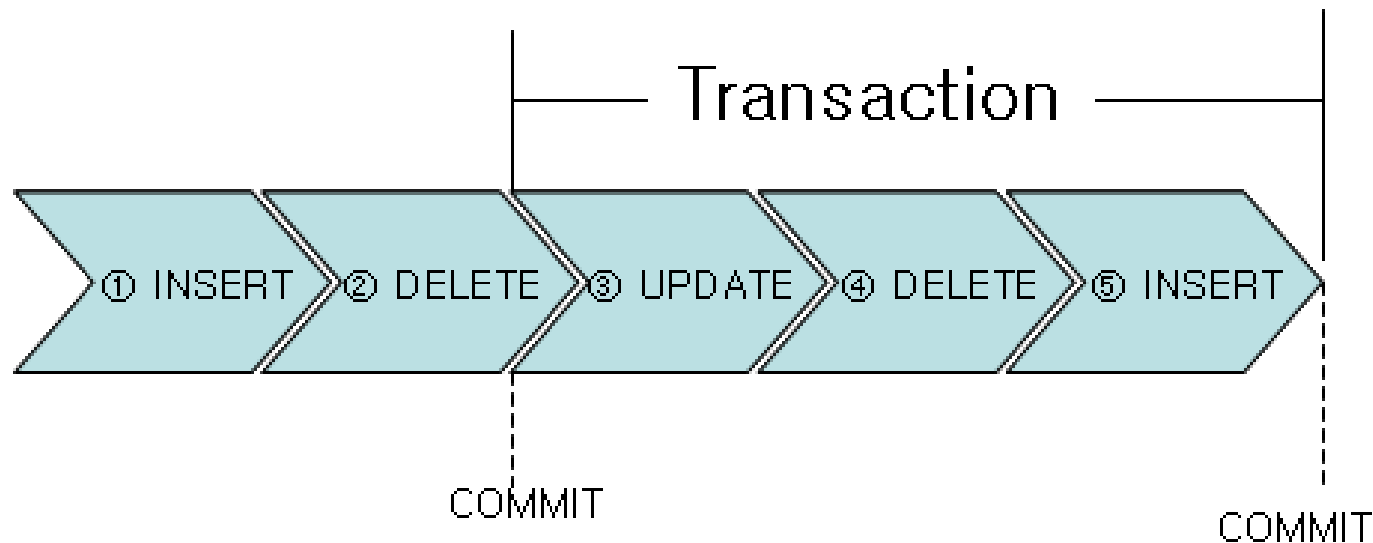


- ❖ 여러 개의 DML 명령어들을 어떻게 하나의 논리적인 단위인 트랜잭션으로 묶을 수 있을까?
- ❖ 트랜잭션은 마지막으로 실행한 커밋(혹은 롤백) 명령 이후부터 새로운 커밋(혹은 롤백) 명령을 실행하는 시점까지 수행된 모든 DML 명령들을 의미합니다.

02. COMMIT과 ROLLBACK



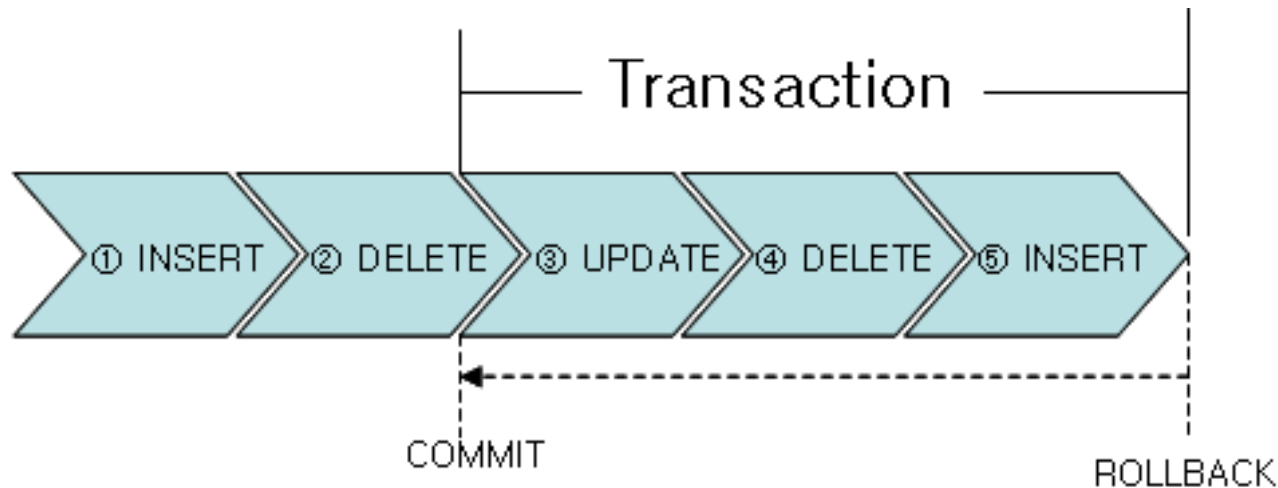
- ❖ 아래 그림에서 UPDATE 문으로 데이터를 갱신하고(③), DELETE 문으로 데이터를 삭제하고(④), INSERT 문을 사용해 데이터를 삽입(⑤)합니다.
- ❖ 만약 이 모든 과정이 오류 없이 수행되었다면 지금까지 실행한 모든 작업(③, ④, ⑤)을 "데이터베이스에 영구 저장하라"는 명령으로 커밋을 수행합니다.



02. COMMIT과 ROLLBACK



- ❖ 롤백 명령은 마지막으로 수행한 커밋 명령까지만 정상 처리(①, ②)된 상태로 유지하고 그 이후에 수행했던 모든 DML 명령어 작업(③, ④, ⑤)들을 취소시켜 이전 상태로 원상 복귀시킵니다.



- ❖ 트랜잭션은 이렇듯 All-OR-Nothing 방식으로 DML 명령어들을 처리합니다.

02. COMMIT과 ROLLBACK



- ❖ COMMIT과 ROLLBACK은 다음과 같은 장점이 있습니다.
- ❖ COMMIT 명령어과 ROLLBACK 명령어의 장점
 - 데이터 무결성이 보장됩니다.
 - 영구적인 변경 전에 데이터의 변경 사항을 확인할 수 있습니다.
 - 논리적으로 연관된 작업을 그룹화할 수 있습니다.

02. COMMIT과 ROLLBACK



❖ 이번에는 COMMIT과 ROLLBACK 명령어를 정리 해보도록 하자.

❖ COMMIT 명령어

- Transaction(INSERT, UPDATE, DELETE) 작업 내용을 실제 DB에 저장합니다.
- 이전 데이터가 완전히 UPDATE 됩니다.
- 모든 사용자가 변경된 데이터의 결과를 볼 수 있습니다.

❖ ROLLBACK 명령어

- Transaction(INSERT, UPDATE, DELETE) 작업 내용을 취소합니다.
- 이전 COMMIT한 곳 까지만 복구합니다.

02. COMMIT과 ROLLBACK



- ❖ 데이터베이스 사용자가 COMMIT이나 ROLLBACK 명령어를 명시적으로 수행시키지 않더라도 다음과 같은 경우에 자동 커밋 혹은 자동 롤백이 발생합니다.
- ❖ 자동 COMMIT 명령과 자동 ROLLBACK 명령이 되는 경우
 - SQL* PLUS가 정상 종료되었다면 자동으로 COMMIT되지만, 비정상 종료되었다면 자동으로 ROLLBACK 합니다.
 - DDL과 DCL 명령문이 수행된 경우 자동으로 COMMIT 됩니다.
 - 정전이 발생했거나 컴퓨터 Down시(컴퓨터의 전원이 끊긴) 자동으로 ROLLBACK 됩니다.

〈실습하기〉 롤백으로 이전으로 복구하기

부서번호가 10번인 부서에 대해서만 삭제하려고 했는데 테이블 내의 모든 로우가 삭제되어 아무런 데이터도 찾을 수 없게 되었다더라도 ROLLBACK 문을 사용하여 이전 상태로 되돌릴 수 있습니다.

1. DELETE 문으로 테이블을 삭제합니다.

```
DELETE FROM DEPT01;
```

2. 만일 부서번호가 20번인 부서에 대해서만 삭제하려고 했는데 위와 같은 명령을 수행했다면 테이블 내의 모든 로우가 삭제되어 다음과 같이 아무런 데이터도 찾을 수 없게 됩니다. 이전 상태로 되돌리기 위해서 ROLLBACK 문을 수행합니다.

```
ROLLBACK;
```

〈실습하기〉 커밋으로 삭제 영구 저장하기

원래하려고 했던 부서번호가 20번인 부서만 삭제해 봅시다.

1. 이번에는 부서번호 20번 사원에 대한 정보만 삭제한 후, 확인합니다.

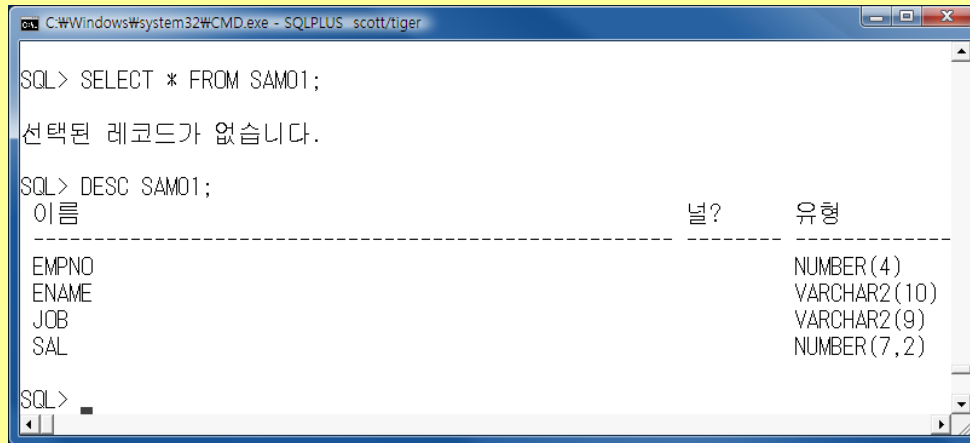
```
DELETE FROM DEPT01 WHERE DEPTNO=20;
```

2. 데이터를 삭제한 결과를 물리적으로 영구히 저장하기 위해서 커밋을 수행합니다.

```
COMMIT;
```


<탄탄히 다지기>

1. 서브 쿼리문을 이용하여 다음과 같은 구조로 SAM01 테이블을 생성하시오.
존재할 경우 DROP TABLE로 삭제 후 생성하시오.



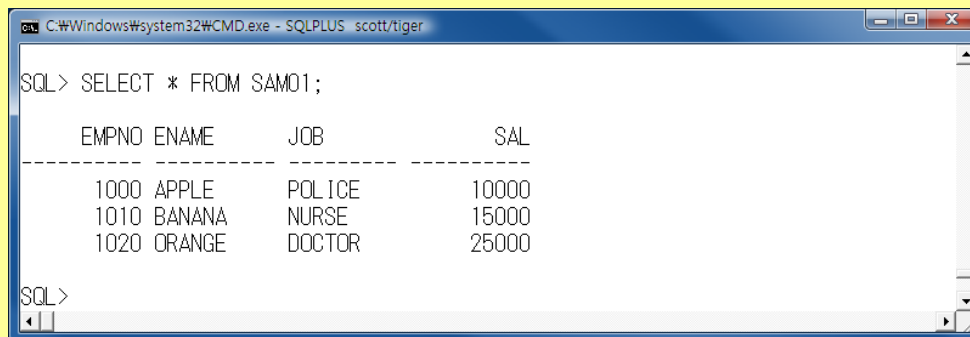
The screenshot shows a SQLPLUS window with the following commands and output:

```
SQL> SELECT * FROM SAM01;

선택된 레코드가 없습니다.

SQL> DESC SAM01;
이름                널?       유형
-----
EMPNO                NUMBER(4)
ENAME                VARCHAR2(10)
JOB                  VARCHAR2(9)
SAL                  NUMBER(7,2)
```

2. SAM01 테이블에 다음과 같은 데이터를 추가하시오.



The screenshot shows a SQLPLUS window with the following command and output:

```
SQL> SELECT * FROM SAM01;
```

EMPNO	ENAME	JOB	SAL
1000	APPLE	POLICE	10000
1010	BANANA	NURSE	15000
1020	ORANGE	DOCTOR	25000

1.3 NULL 값 삽입하는 다양한 방법



- ❖ 데이터를 입력하는 시점에서 해당 컬럼 값을 모르거나 확정되지 않았을 경우에는 NULL 값을 입력해야 합니다.
- ❖ NULL 값 삽입은 암시적인 방법과 명시적인 방법이 있습니다.
- ❖ 암시적 방법은 컬럼 명 리스트에 컬럼을 생략하는 것입니다. 즉, 다른 컬럼은 값을 입력하지만 이렇게 생략한 컬럼에는 암시적으로 NULL 값이 할당되는 것입니다.
- ❖ 명시적 방법은 VALUES 리스트에 명시적으로 NULL을 입력합니다.

1.3 NULL 값 삽입하는 다양한 방법



- ❖ 부서 테이블에 컬럼이 NULL 값을 허용하는지 살펴보기 위해서 DESC 명령을 실행합니다.
- ❖ DEPT 테이블의 DEPTNO 컬럼은 NOT NULL 제약조건이 지정되어 있습니다.

```
C:\Windows\system32\CMD.exe - SQLPLUS scott/tiger

SQL> DESC DEPT
이름                널?       유형
-----
DEPTNO              NOT NULL  NUMBER(2)
DNAME               VARCHAR2(14)
LOC                 VARCHAR2(13)

SQL>
```

- ❖ NOT NULL 제약조건이 지정된 DEPTNO 컬럼은 널 값을 입력하지 못합니다.
- ❖ 오라클이 제공해 주는 DEPT 테이블의 DEPTNO 컬럼에 널값을 허용하지 못하도록 오라클 내부에서 이미 컬럼에 제약조건을 지정해 놓은 상태입니다.
- ❖ 컬럼에 널값을 허용하지 못하도록 하려면 컬럼에 제약조건을 지정해야 합니다.

암시적으로 NULL 값의 삽입



- ❖ 다음은 지역명이 결정되지 않은 30번 부서에 부서명만 입력하려고 합니다.
- ❖ 저장할 값을 명확하게 알고 있는 컬럼 명만 명시적으로 기술한 후에 그에 매칭되는 값을 VALUES 절 다음에 기술합니다.

```
INSERT INTO DEPT01  
(DEPTNO, DNAME)  
VALUES (30, 'SALES');
```

명시적으로 NULL 값의 삽입



- ❖ 컬럼명을 명시적으로 기술하지 않으면 테이블이 갖고 있는 모든 컬럼에 값을 지정해야 합니다.

```
INSERT INTO DEPT01  
VALUES (40, 'OPERATIONS', NULL);
```

- ❖ 지역명이 결정되어 지지 않았더라도 반드시 값을 3개 지정해야 하기 때문에 명시적으로 VALUES 리스트에서 지역명에 NULL을 입력해야 합니다.

명시적으로 NULL 값의 삽입

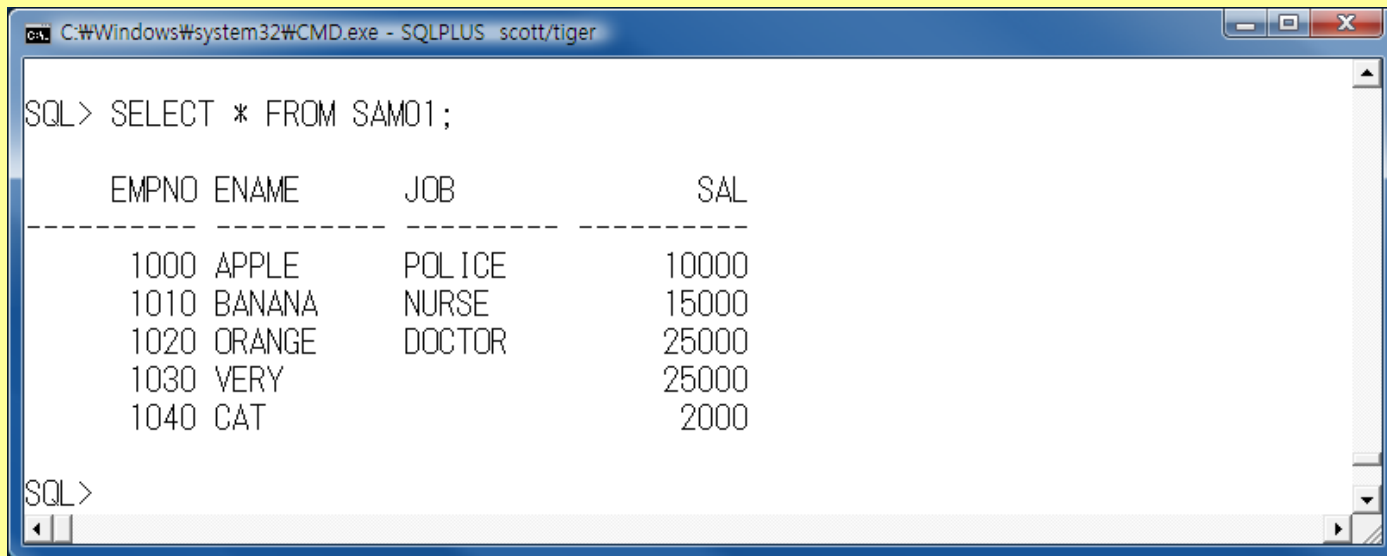


- ❖ NULL 값을 갖는 칼럼을 추가하기 위해서 NULL 대신 ''를 사용할 수 있습니다.
- ❖ 이번에는 지역명이 아닌 부서명이 결정되지 않아 부서명에 NULL 값을 입력한 예입니다.

```
INSERT INTO DEPT01  
VALUES (50, '', 'CHICAGO');
```

<탄탄히 다지기>

3. 문제 1에서 생성한 SAM01 테이블에 다음과 같이 NULL 값을 갖는 행을 추가하시오.



The screenshot shows a Windows command prompt window titled "C:\Windows\system32\CMD.exe - SQLPLUS scott/tiger". The user has entered the SQL command "SELECT * FROM SAM01;". The output displays a table with four columns: EMPNO, ENAME, JOB, and SAL. The data rows are as follows:

EMPNO	ENAME	JOB	SAL
1000	APPLE	POLICE	10000
1010	BANANA	NURSE	15000
1020	ORANGE	DOCTOR	25000
1030	VERY		25000
1040	CAT		2000

The command prompt shows the prompt "SQL>" at the bottom left.

1.4 서브 쿼리로 데이터 삽입하기



- ❖ INSERT INTO 다음에 VALUES 절을 사용하는 대신에 서브 쿼리를 사용할 수 있습니다.
- ❖ 이렇게 하면 기존의 테이블에 있던 여러 행을 복사해서 다른 테이블에 삽입할 수 있습니다.
- ❖ 이 때 주의할 점은 INSERT 명령문에서 지정한 컬럼의 개수나 데이터 타입이 서브 쿼리를 수행한 결과와 동일해야 한다는 점입니다.

〈실습하기〉 서브 쿼리로 데이터 삽입하는 예제

1. 서브 쿼리로 데이터 삽입하기 위해서 우선 테이블을 생성하되 데이터는 복사하지 않고 빈 테이블만 생성합니다.

```
DROP TABLE DEPT02;  
CREATE TABLE DEPT02  
AS  
SELECT * FROM DEPT WHERE 1=0;
```

2. 테이블 구조만을 복사해서 내용을 갖지 않는 테이블에 서브 쿼리로 로우를 입력해 봅니다.

```
INSERT INTO DEPT02  
SELECT * FROM DEPT;
```

〈탄탄히 다지기〉

- 1.1. 트랜잭션 내용을 실제 DB에 저장하기 위해서는 () 문을 사용한다.
- 1.2. 트랜잭션 내용을 취소하기 위해서는 () 문을 사용하여 이전 () 한 부분까지 변경 전 데이터로 복구한다.

03. 자동 커밋



- ❖ DDL 문에는 CREATE, ALTER, DROP, RENAME, TRUNCATE 등이 있습니다.
- ❖ 이러한 DDL문은 자동으로 커밋(AUTO COMMIT)이 발생합니다.

〈실습하기〉 CREATE문에 의한 자동 커밋

CREATE문에 의한 자동 커밋에 의해서 이전에 수행했던 DML 명령어가 자동 커밋됨을 확인해 봅시다.

1. 부서 번호가 40번인 부서를 삭제합니다.

```
DELETE * FROM dept02  
WHERE deptno=40;
```

2. 삭제 후 부서 테이블(DEPT)과 동일한 내용을 갖는 새로운 테이블(DEPT03)을 생성합니다.

```
CREATE TABLE DEPT03  
AS  
SELECT * FROM DEPT;
```

3. DEPT02 테이블의 부서번호가 40번인 부서를 다시 되살리기 위해서 ROLLBACK 명령문을 수행하여도 이미 수행한 CREATE 문 때문에 자동으로 커밋이 발생하였으므로 되살릴 수 없습니다.

〈실습하기〉 TRUNCATE 문 실패에 의한 자동 커밋

TRUNCATE 문이 실패되더라도 자동 커밋되어 이전에 수행했던 DML 명령어가 자동 커밋됨을 확인해 봅시다

1. 부서 테이블(DEPT03)에서 부서 번호가 20번인 부서를 삭제합니다.

```
DELETE FROM DEPT03  
WHERE DEPTNO=20;
```

2. TRUNCATE 문을 실행시키되 테이블 명을 일부러 잘못 적어서 에러를 유도합니다.

```
TRUNCATE TABLE DEPTPPP;
```

3. 부서번호가 20번인 부서를 다시 되살리기 위해서 ROLLBACK 명령문을 수행하여도 TRUNCATE 문이 수행되면서 자동으로 커밋이 발생하였으므로 되살릴 수 없습니다.

04. 트랜잭션을 작게 분할하는 SAVEPOINT

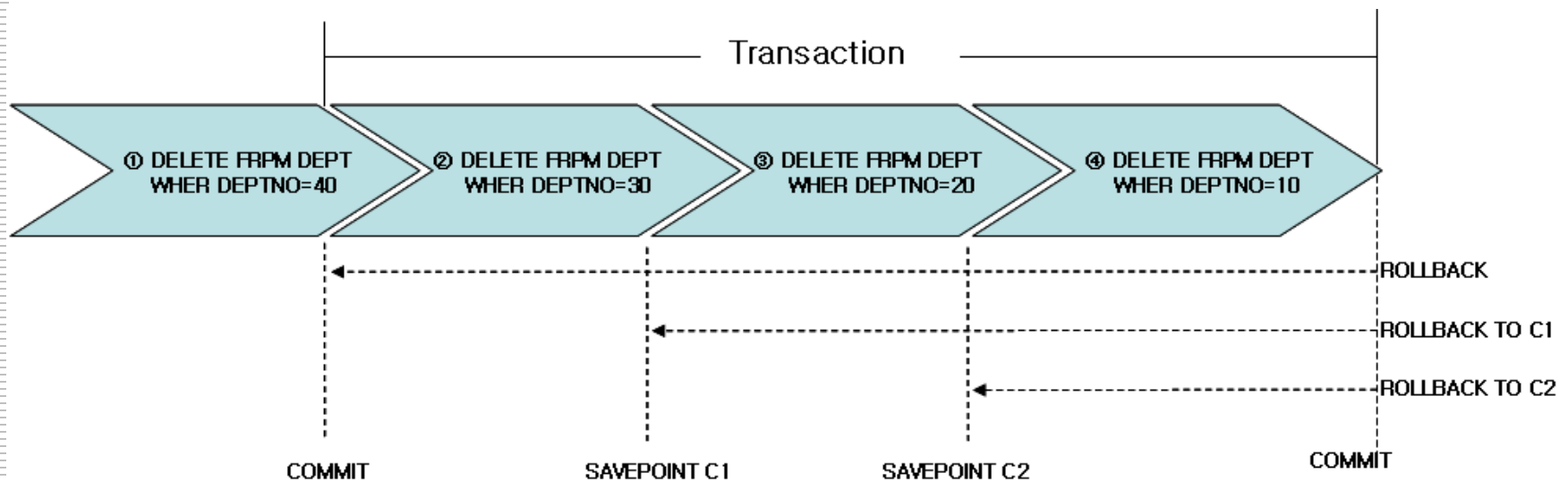


- ❖ **SAVEPOINT 명령을 써서 현재의 트랜잭션을 작게 분할할 수 있습니다.**
- ❖ **저장된 SAVEPOINT는 ROLLBACK TO SAVEPOINT 문을 사용하여 표시한 곳까지 ROLLBACK할 수 있습니다.**
- ❖ **여러 개의 SQL 문의 실행을 수반하는 트랜잭션의 경우, 사용자가 트랜잭션 중간 단계에서 세이프포인트를 지정할 수 있습니다.**
- ❖ **이 세이프포인트는 차후 롤백과 함께 사용해서 현재 트랜잭션 내의 특정 세이프포인트까지 롤백할 수 있게 됩니다.**

04. 트랜잭션을 작게 분할하는 SAVEPOINT



- ❖ 아래 그림을 보면 COMMIT 명령이 내려진 후 다음 COMMIT 명령이 나타날 때까지가 하나의 트랜잭션으로 구성되므로 ②번에서 ④번까지가 하나의 트랜잭션이 됩니다.
- ❖ 이렇게 트랜잭션을 구성할 때 중간 중간 SAVEPOINT 명령으로 위치를 지정해 놓으면(예를 들어 C) 하나의 트랜잭션 내에서도 ROLLBACK TO C(SAVEPOINT 문을 사용하여 표시한 곳)까지 ROLLBACK할 수 있습니다.



04. 트랜잭션을 작게 분할하는 SAVEPOINT



- ❖ 다음은 SAVEPOINT로 특정 위치를 지정하기 위한 사용 형식입니다.

```
SAVEPOINT LABEL_NAME;
```

- ❖ SAVEPOINT로 지정해 놓은 특정 위치로 되돌아가기 위한 사용 형식입니다.

```
ROLLBACK TO LABEL_NAME;
```


〈실습하기〉 트랜잭션 중간 단계에서 세이브포인트 지정하기

다음과 같이 트랜잭션 중간 단계에서 세이브포인트를 지정해 보도록 하겠습니다.

40번 부서 삭제



COMMIT

30번 부서 삭제



세이브포인트 C1 설정

20번 부서 삭제



세이브포인트 C2 설정

10번 부서 삭제

〈실습하기〉 트랜잭션 중간 단계에서 세이프포인트 지정하기

1. 부서번호가 40번인 부서를 삭제한 후에 커밋을 수행하여 새롭게 트랜잭션을 시작합니다.

```
DELETE FROM DEPT01 WHERE DEPTNO=40;  
COMMIT;
```

2. 이번엔 부서번호가 30번인 부서를 삭제합니다.

```
DELETE FROM DEPT01 WHERE DEPTNO=30;
```

3. 세이프포인트 C1를 설정한 후, 부서번호가 20번인 사원을 삭제합니다.

```
SAVEPOINT C1;  
DELETE FROM DEPT01 WHERE DEPTNO =20;
```

4. 세이프포인트 C2를 설정한 후, 부서번호가 10번인 사원을 삭제합니다.

```
SAVEPOINT C2;  
DELETE FROM DEPT01 WHERE DEPTNO =10;
```

〈실습하기〉 트랜잭션 중간 단계로 되돌리기

이제 부서번호가 10번인 사원을 삭제하기 바로 전으로 되돌리려면 어떻게 해야 할까요? 세이브 포인트를 이용해서 트랜잭션 중간 단계로 되돌려 봅시다.

1. 지금 ROLLBACK 명령을 내리게 된다면 이전 COMMIT 지점으로 되돌아가므로 10, 20, 30번 부서의 삭제가 모두 취소됩니다. 따라서 원했던 10번 부서의 삭제 이전까지만 되돌리려면 다시 30, 20번의 부서를 삭제해 주어야 할 것입니다.

```
ROLLBACK TO C2;
```

2. 위 결과 화면을 보면 세이브포인트 C2 지점으로 이동되어 10번 부서의 삭제 이전으로 되돌려진 것을 확인할 수 있습니다.

```
ROLLBACK TO C1;
```

3. 마지막으로 이전 트랜잭션까지 롤백한 후의 결과를 봅시다.

```
ROLLBACK;
```



Thank You !

Dynamic_오라클 11g + PL/SQL 입문 11장