I. Transaction

#Commit, Rollback 실습

```
SQL> SELECT * FROM emp;
SQL> UPDATE emp SET sal = 0 WHERE ename LIKE 'A%';
SQL> SELECT * FROM emp;
SQL> DELETE FROM emp WHERE deptno = 10;
SQL> SELECT * FROM emp;
```

 # 변경을 취소하여보자

```
SQL> ROLLBACK;
SQL> SELECT * FROM emp;
```

 # 실습을 위해 emp를 복제하자.

```
SQL> CREATE TABLE emp2 AS (SELECT * FROM emp);

SQL> SELECT * FROM emp2;

SQL> update emp2 SET sal = 6000 WHERE ename='SCOTT'
SQL> SELECT * FROM emp2;
SQL> DELETE FROM emp2 WHERE empno < 7700;
SQL> SELECT * FROM emp2;
SQL> INSERT INTO emp2 VALUES (100, 'LEE', 'SI', NULL, SYSDATE, 9999, 0, NULL);
```

 # 변경을 commit

```
SQL> COMMIT;
SQL> SELECT * FROM emp2;
```

 # 혹시 취소가 되나?

```
SQL> ROLLBACK;
SQL> SELECT * FROM emp2;
```

 # 한번 commit된 변경은 취소가 되지 않는다.

 # Delete & Truncate

 # Delete는 Rollback이 되나?

```
SQL> DELETE FROM emp2;
```

SQL> SELECT * FROM emp2;
SQL> Rollback;
SQL> SELECT * FROM emp2;

 # Truncate는 Rollback이 되나?
SQL> TRUNCATE TABLE emp2;
SQL> SELECT * FROM emp2;
SQL> Rollback;
SQL> SELECT * FROM emp2;

SAVEPOINT

 # 데이터 로드
SQL> INSERT INTO emp2 (SELECT * FROM emp);
SQL> SELECT * FROM emp2;

 # commit
SQL> COMMIT;
SQL> SELECT * FROM emp2;

 # 변경 & 저장점 생성
SQL> INSERT INTO emp2 VALUES (100, 'LEE', 'SI', NULL, SYSDATE, 9999, 0, NULL);
SQL> SAVEPOINT a;
SQL> SELECT * FROM emp2;

 # 변경 & 저장점 생성
SQL> UPDATE emp2 SET sal = sal * 2 WHERE job = 'SALESMAN';
SQL> SAVEPOINT b;
SQL> SELECT * FROM emp2;

 # 변경
SQL> DELETE FROM emp2 WHERE sal > 3000;
SQL> SELECT * FROM emp2;

 # 특정 저장점으로 rollback;
SQL> ROLLBACK TO b;
SQL> SELECT * FROM emp2;

 # 특정 저장점으로 rollback;

SQL> ROLLBACK TO a;

SQL> SELECT * FROM emp2;

 # 전체 rollback;

SQL> ROLLBACK;

SQL> SELECT * FROM emp2;

SQL> ROLLBACK TO a;

(에러 발생 이유는 무엇인가?)

Read Consistency

 # 동시에 두개의 SQL*PLUS를 실행하여 각각 아래의 SQL을 순서에 따라 수행해보면서 Lock의 효과를 확인해보자.

 # emp2가 없으면 앞 페이지의 SQL문을 통하여 emp2를 만들자

SQL*Plus 1	SQL*Plus 2
SQL> conn scott SQL> SELECT * FROM emp2; **#변경** SQL> INSERT INTO emp2(empno, ename) VALUES (10, 'KIM'); SQL> SELECT * FROM emp2; **# 'KIM'이 보이는가?** **# 변경이 보이는가?**	 SQL> conn scott SQL> SELECT * FROM emp2; SQL> SELECT * FROM emp2; **# 'KIM'이 보이는가?** **# commit되지 않은 트랜잭션의 변경은 다른 세션에서는 보이지 않는다.** **# 변경** SQL> UPDATE emp2 SET sal = sal * 10;

SQL> SELECT * FROM emp2; **# commit** SQL> commit;	 **#변경이 보이는가?** SQL> SELECT * FROM emp2; **# Rollback** SQL> rollback; SQL> SELECT * FROM emp2; **# 어떤 변경만 반영되었나?**

Lock

# 변경 SQL> UPDATE emp2 SET sal = 10000 WHERE ename = 'SCOTT'; **# commit하면 모든 lock 해제** SQL> commit; SQL> SELECT * FROM emp2;	 **# 변경 시도 (Lock이 안걸린 row)** SQL> UPDATE emp2 SET sal = 20000 WHERE ename='SMITH'; **# 변경 시도 (Lock이 걸린 row)** SQL> UPDATE emp2 SET job = 'MANAGER' WHERE ename='SCOTT'; **# SCOTT은 이미 Session 1에서 Lock을 건** **# Row이므로, Lock을 얻지 못해** **#실행이 멈추어있다.** **# Session 1에서 commit을 하면 변경 시도가 완료된다.** SQL> SELECT * FROM emp2;

# 변경 시도 (lock이 걸린 row) SQL> UPDATE emp2 SET sal = 10000 WHERE ename = 'SMITH';	
	# 여기서 rollback을 하면 모든 lock 해제 SQL> ROLLBACK;
# 현재 트랜잭션이 잡고 있는 Lock은?	**# 현재 트랜잭션이 잡고 있는 Lock은?**
# 데이터 다시 로드 SQL> DELETE FROM emp2; SQL > INSERT INTO emp2 (SELECT * FROM emp); SQL> commit;	

Non-Repeatable Read: 하나의 Transaction 안에서 동일한 질의 결과가 다르게 나타날 수 있다. Phantom Read 문제가 발생할 수 있음. (**Statement-level Read Consistency이기 때문에 발생)**

SQL> UPDATE emp2 SET comm = (SELECT count(*) FROM emp2) WHERE ename='SMITH'; SQL> SELECT count(*) FROM emp2;	
	# 새로운 row 추가 (phantom) SQL> INSERT INTO emp2(empno, ename) VALUES (20, 'LEE'); SQL> commit;
# count 결과가 달라진다. SQL> SELECT count(*) FROM emp2; SQL> UPDATE emp2 SET comm = (SELECT count(*) FROM emp2) WHERE ename='ALLEN'; SQL> SELECT * FROM emp2; **# 분명 같은 SQL을 SMITH와 ALLEN에게 수행했는데 결과가 다르게 나타난다.** SQL> Rollback;	
	# 데이터 다시 로드 SQL> DELETE FROM emp2; SQL > INSERT INTO emp2 (SELECT * FROM emp); SQL> commit;

Read-Only Transaction: 다른 트랜잭션이 commit한 내용이 중간에 보이지 않음
Transaction-level Read Consistency 제공. 대신 변경 연산은 수행할 수 없다.

# Read-Only로 설정 SQL> SET TRANSACTION READ ONLY; SQL> UPDATE emp2 SET sal = 100; **(오류발생!)**	

SQL> SELECT * FROM emp2; **# 변경이 보이는가?** SQL> SELECT * FROM emp2; SQL> SELECT count(*) FROM emp2; SQL> commit; **# 변경이 보이는가?** SQL> SELECT * FROM emp2; SQL> SELECT count(*) FROM emp2; **# 데이터 다시 로드** SQL> DELETE FROM emp2; SQL > INSERT INTO emp2 (SELECT * FROM emp); SQL> commit;	**#다른 트랜잭션에서 변경을 해보자.** SQL> DELETE FROM emp2 WHERE sal>1000; SQL> commit; SQL> SELECT * FROM emp2;

SET TRANSACTION ISOLATION LEVEL SERIALIZABLE:

#트랜잭션 isolation 레벨 설정 SQL> SET TRANSACTION ISOLATION LEVEL SERIALIZABLE; SQL> SELECT * FROM emp2; SQL> SELECT * FROM emp2;	**# 변경 (commit완료)** SQL> SELECT * FROM emp2; SQL> UPDATE emp2 SET sal = 10000 WHERE ename='SMITH'; SQL> COMMIT; SQL> SELECT * FROM emp2;

# 아래는 수행 가능한가? SQL> UPDATE emp2 SET sal = 0 WHERE ename= 'ALLEN'; **#아래는 수행가능한가?** SQL> UPDATE emp2 SET sal = 0 WHERE ename = 'SMITH'; **#원인은 무엇인가?** SQL> ROLLBACK; SQL> SET TRANSACTION ISOLATION LEVEL READ COMMITED; **# 데이터 다시 로드** SQL> DELETE FROM emp2; SQL > INSERT INTO emp2 (SELECT * FROM emp); SQL> commit;	

DEADLOCK: 서로 상대방의 lock을 기다리고 있는 교착상태에 빠지는 경우
Oracle이 주기적으로 deadlock을 검사하여 교착상태를 해결한다.
(한쪽 transaction을 강제로 에러 처리)

# Allen lock 획득 SQL> UPDATE emp2 SET sal = 0 WHERE ename= 'ALLEN';	
	# smith lock획득 SQL> UPDATE emp2 SET sal = 0 WHERE ename= 'SMITH';
# smith lock 시도 (wait) SQL> UPDATE emp2 SET sal = 0 WHERE ename= 'SMITH';	
	# allen lock 시도 (wait) SQL> UPDATE emp2 SET sal = 0 WHERE ename= 'ALLEN';
# 둘중 어느 트랜잭션도 더 이상 진행할 수 없는 DEADLOCK 상태에 빠졌다. 잠시 기다리면 둘중 하나의 트랜잭션에서 에러가 발생한다. **#여기서 ROLLBACK이나 COMMIT을 수행하여 모든 Lock을 반환하면 상대 트랜잭션이 다시 진행된다.**	