REM Author :
REM Date :
REM Objective : Chapter 8. TCL
REM Environment : Ubuntu Server 20.04 LTS, HeidiSQL 10.2.0, MySQL Community Server 5.7.34.0

REM **Transaction**

1. **Transaction** 이란?
   1)한 개 이상의 실행될 **SQL** 문장의 집합
   2)**Database**의 논리적 연산단위.
   3)밀접히 관련되어 분리될 수 없는 한 개 이상의 **Database** 조작.
   4)하나의 **Transaction**에는 하나 이상의 **SQL** 문장이 포함된다.
   5)분할할 수 없는 최소의 단위.
   6)데이터의 일관성을 보장
   7)행의 **LOCK** 처리를 기본으로 함
   8)데이터의 변경시 융통성을 제공하고, 사용자의 프로세스가 예기치 않게 중단되거나, 시스템 장애가 발생하여 데이터의 일관성이 어렵게 된 경우에도 일관성을 보장하기 위한 시스템적 응용
   9)전부 적용하거나 전부 취소한다.
   10)즉 **ALL OR** NOTHING의 개념.
   11)Oracle에서는 **Transaction**의 대상이 되는 **SQL** 문장을 실행하면 자동으로 시작되고, **COMMIT**또는 **ROLLBACK**을 실행한 시점에서 종료된다.

2. **Transaction**의 특성
   1)원자성(Atomicity) : **Transaction**에서 정의된 연사들은 모두 성공적으로 실행되던지 아니면 전혀 실행되지 않은 상태로 남아 있어야 한다.(**all or** nothing)
   2)일관성(Conststency) : **Transaction**이 실행되기 전의 **Database** 내용이 잘못 되어 있지 않다면 **Transaction**이 실행된 이후에도 **Database**의 내용에 잘못이 있으면 안 된다.
   3)고립성(**Isolation**) : **Transaction**이 실행되는 도중에 다른 **Transaction**의 영향을 받아 잘못된 결과를 만들어서는 안된다.
   4)지속성(Durability) : **Transaction**이 성공적으로 수행되면 그 **Transaction**이 갱신한 **Database**의 내용은 영구적으로 저장된다.

3.TCL(Tracnsaction Control **Language**)
   1)**COMMIT** : 보류중인 데이터를 영구적인 데이타베이스로 변경사항을 저장하고 현재의 **TRANSACTION** 을 종료한다.
   2)**ROLLBACK [TO SAVEPOINT** name]: 보류중인 데이터의 변경사항을 모두 되돌리고 현재의 트랜젝션을 종료한다. 만일 특정지점을 지정하지 않으면 모든 트랜젝션 취소한다.
   3)**SAVEPOINT** name :현재의 트랜젝션의 저장점을 표시한다.

4.**Transaction** 의 범위
   -아래의 사항들은 자동으로 **TRANSACTION** 이 적용된다.
   1)DDL 문이 실행된 경우
   2)DCL 문이 실행된 경우
   3)명시적으로 **commit** 이나 **rollback** 이 실행되지 않은 상태에서 **exit** 를 해서 sqlplus 를 종료할 때
   4)강제적으로 **Transaction** 이 적용되는 경우
   5)시스템 장애가 발생할 경우

5.트랜잭션 암시적 처리
   1)자동 **COMMIT**은 아래의 사항에 발생
      -DDL 문이 실행된 경우
      -DCL 문이 실행된 경우
      -명시적으로 **COMMIT** 또는 **ROLLBACK** 이 실행되지 않은 채 **SQL***Plus 에서 정상 종료(**exit**)한 경우
   2)-자동 **ROLLBACK**
      -**SQL***Plus 의 비정상 종료 시 또는 시스템 장애 발생시

6. **COMMIT**이나 **ROLLBACK** 이전의 **Data**의 상태
   1)단지 Memory Buffer에만 영향을 받았기 때문에 **Data**의 변경 이전 상태로 복구 가능하다.
   2)현재 사용자는 **SELECT** 문장으로 결과를 확인 가능하다.
   3)다른 사용자는 현재 사용자가 수행한 명령의 결과를 볼 수 없다.
   4)변경된 행은 잠금(LOCKING)이 설정되어서 다른 사용자가 변경할 수 없다.

REM **COMMIT**
1. 변경 사항을 영구히 저장

```
UPDATE emp SET deptno = 10 WHERE empno = 7782;
COMMIT;
```

2. **COMMIT** 이후의 **Data**의 상태
   1)**Data**에 대한 변경 사항이 **Database**에 반영된다.
   2)이전 **Data**는 영원히 잃어버리게 된다.

3)모든 사용자는 결과를 볼 수 있다.
4)관련된 행에 대한 잠금(LOCKING)이 풀리고, 다른 사용자들이 행을 조작할 수 있게 된다.

REM **ROLLBACK**
1. 보류 중인 변경 내용을 모두 되돌림
   **-Data** 변경 사항이 취소되어 **Data**의 이전 상태로 복구되며, 관련된 행에 대한 잠금(LOCKING)이 풀리고 다른 사용자들이 **Data**의 변경을 할 수 있게 된다.

2. **ROLLBACK**이후의 상태
   1)**Data**에 대한 변경 사항은 취소된다.
   2)이전 **Data**는 다시 재저장된다.
   3)관련된 행에 대한 잠금(LOCKING)이 풀리고, 다른 사용자들이 행을 조작할 수 있게 된다.

```
DELETE FROM emp;
ROLLBACK;
```

REM **COMMIT**과 **ROLLBACK**의 효과
1. **Data** 무결성 보장
2. 영구적인 변경을 하기 전에 **Data**이 변경 사항 확인 가능
3. 논리적으로 연관된 작업을 그룹핑하여 처리 가능.

REM **SAVEPOINT**
1. 현 시점에서 **SAVEPOINT**까지 **Transaction**의 일부만 **ROLLBACK** 할 수 있다.
2. 따라서 복잡한 대규모 **Transaction**에서 Error가 발생했을 때 **SAVEPOINT**까지의 **Transaction**만 **Rollback**하고 실패한 부분에 대해서만 다시 실행할 수 있다.
3. 복수의 저장점을 정의할 수 있으며, 동일이름으로 저장점을 정의했을 때는 나중에 정의한 저장점이 유효하다.
4. Syntax
   **SAVEPOINT** savepoint_name;
5. 저장점까지 **rollback**할 때는 **ROLLBACK** 뒤에 저장점 명을 지정한다.
   **ROLLBACK TO** savepoint_name;
6. **Rollback**에 **SAVEPOINT** 명을 부여하여 실행하면 저장점 설정 이후에 있었던 **Data** 변경에 대해서만 원래 **Data** 상태로 되돌아가게 된다.
7. 저장점 지정없이 **ROLLBACK**을 실행했을 경우에는 반영안된 모든 변경 사항을 취소하고 **Transaction** 시작 위치로 되돌아간다.

REM 저장점까지 변경 내용 **ROLLBACK**
1. **SAVEPOINT** 사용하여 현재 트랜잭션에 표시자를 생성하여 트랜잭션을 더 작은 부분으로 나눈 후 **ROLLBACK TO SAVEPOINT** 문을 사용해 보류 중인 변경 내용을 해당 표시자까지 되돌림.

```
START TRANSACTION;

UPDATE emp SET deptno = 10 WHERE empno = 7782;
SAVEPOINT a;

INSERT INTO emp(empno, ename, job, mgr, hiredate, sal, comm, deptno)
VALUES (7999, 'TOM', 'SALESMAN', 7782, CURDATE(), 2000, 2000, 10);

ROLLBACK TO a;
```