08강 데이터 조작과 트랜젝션

DML

- DML(데이터 조작어) – 데이터를 추가하고, 삭제하고 변경하는 명령어 쿼리의 집합체

  - 데이터 추가 : insert
  - 데이터 변경 : update
  - 데이터 삭제 : delete

데이터 추가 - INSERT

- Insert는 테이블에 데이터를 삽입할 때 사용하는 명령어이다.

- 기본 사용법
  - insert into 테이블명 (컬럼명1,컬럼명2…) values (데이터1, 데이터2…);

  테이블의 모든 컬럼을 채우면 컬럼명을 생략할 수도 있다.
  - insert into 테이블명 values (데이터1, 데이터2…);
  이때 데이터 순서는 테이블에 명시된 순서여야 한다.

데이터 추가 - INSERT

- 예제 : 사원 테이블에 데이터를 추가해본다.

```
INSERT INTO employee (eno,ename, job, manager, hiredate, salary, commission, dno)
VALUES(8121,'ALICE','CLERK',7788,sysdate,1200,100,10);
```

```
INSERT INTO employee
VALUES(8231,'KATHERINE' ,'SALESMAN',7698,sysdate,1750,800,30);
```

- 데이터의 영구적인 저장을 위해서 COMMIT를 해주고 조회를 해본다.

```
COMMIT;

SELECT * FROM employee;
```

데이터 추가 - INSERT

- 어떤 값을 넣어야 할지 모를 때 - NULL
  - 컬럼을 생략하고 데이터로 생략하면 NULL값으로 입력이 된다.
  - 명시적으로 NULL이라고 입력해도 NULL값으로 입력이 된다.
  - 테이블 만들 때 NOT NULL 선언하면 NULL 입력 시 에러 발생

```
INSERT INTO employee (eno)
VALUES(8500);

INSERT INTO employee (eno,ename)
VALUES(8600,NULL);

INSERT INTO employee (eno,ename)
VALUES(8700,'');
```

데이터 추가 - INSERT

- 날짜 입력 시 형식에 맞게 입력해야 한다.(YYYY/MM/DD)

```
INSERT INTO employee (eno,ename,hiredate)
VALUES(8800,NULL,'2020/12/31');
```

- TO_DATE함수를 이용할 수도 있다.

```
INSERT INTO employee (eno,ename,hiredate)
VALUES(8900,NULL,TO_DATE('2020-12-31','YYYY-MM-DD'));
```

- 현재 날짜를 이용할 때는 sysdate를 사용한다.

```
INSERT INTO employee (eno,ename,hiredate)
VALUES(8950,NULL,sysdate);
```

데이터 복사

- 서브쿼리를 사용해서 다중 행을 입력할 수 있다.
  - 테스트로 사용할 테이블 복제를 진행한다.

```
CREATE TABLE emp2
AS SELECT * FROM employee WHERE 0=1;


SELECT * FROM emp2;
```

- 다중행 입력 기능으로 원래 테이블의 데이터를 가져온다.

```
INSERT INTO emp2
SELECT * FROM employee;


SELECT * FROM emp2;
```

데이터 변경 - UPDATE

- UPDATE는 기존 데이터를 변경할 때 사용하는 명령어이다.

- 기본 사용법
  - update 테이블명 set 컬럼명1=데이터1 ,컬럼명2=데이터2...where 조건;

  조건절을 생략하면 해당 컬럼의 모든 데이터가 변경된다.

  실습전에 테스트용 테이블을 생성해 본다.

```
CREATE TABLE emp3
AS SELECT * FROM employee;
```

데이터 수정 - UPDATE

- 예제 : eno가 8500 사원 이름을 RUNA로 바꾸고 담당업무를 MANAGER로 바꾸어 본다.

```
UPDATE emp3 SET ename = 'RUNA',job ='MANAGER'
WHERE eno=8500;
```

- 예제 : 모든 사원의 커미션을 500으로 바꾸어 본다.

```
UPDATE emp3 SET commission=500;
```

데이터 수정 - UPDATE

- 서브 쿼리를 이용해서 데이터를 변경할 수도 있다.
  - 예제 : DNO정보가 null인 사원의 급여를 급여등급 1등급 최저급여로 맞추어 변경한다.

```
UPDATE emp3
SET salary=(SELECT losal
            FROM salgrade
            WHERE grade=1)
WHERE DNO IS NULL;
```

  - 예제 : 이름이 없는 사원의 소속을 OPERATIONS로 옮기고 PRESIDENT의 직속 부하직원으로 배치한다.

```
UPDATE emp3
SET dno=(SELECT dno
            FROM department
            WHERE dname='OPERATIONS'),
    manager = (SELECT eno
            FROM emp3
            WHERE job='PRESIDENT')
WHERE ename IS NULL;
```

데이터 삭제 - DELETE

- DELETE는 기존 데이터를 삭제할 때 사용하는 명령어이다.

- 기본 사용법
  - delete from 테이블명 where 조건;

  조건에 해당을 레코드를 삭제한다.
  조건절을 생략하면 모든 레코드가 삭제된다.

  실습전에 테스트용 테이블을 생성해 본다.

```
CREATE TABLE emp4
AS SELECT * FROM employee;
```

데이터 삭제 - DELETE

- 예제 : 사원 이름이 Alice인 사원을 삭제하기
  - select문으로 확인해본다.

```
DELETE FROM emp4
WHERE ename='ALICE';
```

- 예제 : where절 없이 삭제해보기 => 삽입명령으로 복원
  - select문으로 확인해본다.

```
DELETE FROM emp4;

INSERT INTO emp4
SELECT * FROM employee;
```

데이터 삭제 - DELETE

- 서브 쿼리로 나온 값을 기준으로 레코드를 삭제할 수 있다
- 예제 : 부서명이 RESEARCH 소속의 사원 정보를 모두 삭제하세요.

```
DELETE FROM emp4
WHERE dno=(SELECT dno
           FROM department
           WHERE dname='RESEARCH');
```

트랜젝션(Transaction)

- 트랜젝션이란 데이터 처리를 위한 논리적인 작업의 단위를 의미함
- 오라클은 이런 트랜젝션을 기반으로 데이터의 일관성을 보장한다.

- SQL 기본 명령어 중에서 DDL,DCL은 하나의 명령어가 하나의 트랜젝션을 이루고 DML은 하나 이상의 명령으로 트랜젝션을 구성한다.

트랜젝션(Transaction)

- 오라클의 트랜젝션은 데이터의 일관성을 위해서 ALL-or Nothing 방식으로 처리한다.
- 즉, 여러 개의 명령어중 하나만 잘못되어도 모든 명령을 취소시켜서 데이터의 일관성을 유지한다.

- 트랜젝션 관리를 위해 제공하는 명령어는 두가지인데.

COMMIT와 ROLLBACK이다.

트랜젝션(Transaction) - COMMIT

- COMMIT 명령은 모든 작업을 정상 처리 완료하고 처리의 모든 과정을 확정하는 명령이다.

- 모든 트랜젝션의 처리과정을 데이터베이스에 반영하고 변경된 모든 내용을 영구 저장을 한다.

- COMMIT 명령어를 수행하면 하나의 트랜젝션 과정을 종료한다.

트랜젝션(Transaction) - ROLLBACK

- ROLLBACK 명령은 작업중 문제가 발생해서 트랜젝션 처리 과정에 발생한 변경 내용을 취소하는 명령이다.

- ROLLBACK은 트랜젝션으로 인한 하나의 묶음 처리가 시작되기 이전 상태로 되돌린다.

트랜젝션(Transaction)

- 예제를 다루기 위해서 부서정보를 가진 테이블을 복제한다.

```
CREATE TABLE depart2
AS
SELECT * FROM department;

SELECT * FROM depart2;
```

- 이제 실수를 해보자 부서번호 10번을 삭제하려다 모든 내용을 삭제하는 실수를 했다고 가정한다.

```
DELETE FROM depart2;
WHERE dno=10
```

- 모든 데이터를 조회해 보면 아무 것도 나오지 않는다.

```
SELECT * FROM depart2;
```

- 이때 ROLLBACK을 하고 다시 데이터를 조회하면 모든 내용이 복원된 것을 확인 할 수 있다

```
ROLLBACK;

SELECT * FROM depart2;
```

트랜젝션(Transaction)

- 이제 다시 정상적으로 10번 부서를 삭제해본다.

```
DELETE FROM depart2
WHERE dno=10;
```

- 그리고 COMMIT을 한 뒤 데이터가 삭제 되어있는지 조회해보고 다시 BOLLBACK를 한 뒤 데이터를 조회 해본다.

```
SELECT * FROM depart2;

COMMIT;

SELECT * FROM depart2;

ROLLBACK;

SELECT * FROM depart2;
```