15강 프로시저와 함수와 트리거

프로시저

- 프로시저의 개념
  - 기본 PL/SQL에서 사용하는 절차적 프로그래밍은 그때 그때 만들어서 사용해야하는 불편함이 있다.
  - 이런 프로그래밍 코드를 묶음으로 지정하기 위한 것이 프로시저이다.

  - 절차적 코드를 하나의 이름 아래 묶어두면 재사용시 이름만으로 해당 코드를 실행할 수 있다.

프로시저

- 프로시저의 생성
  - 프로시저를 생성하기 위해서는 CREATE명령을 사용한다.

CREATE [OR REPLACE] PROCEDURE 프로시저 이름
IS
    선언부
BEGIN
    실행부
END;

```
CREATE PROCEDURE sp_salary
is
    v_salary employee.ename%type;
BEGIN
    SELECT salary into v_salary
    from employee
    WHERE ename='SCOTT';
    dbms_output.put_line('SCOTT의 급여는'||v_salary);
END;
```

프로시저

- 프로시저의 실행
  - 한번 만들어둔 프로시저는 언제든지 이름만 호출하면 사용이 가능하다.

  - EXCUTE 프로시저 이름;

```
EXECUTE sp_salary;
```

프로시저

- 프로시저의 확인
  - 프로시저가 잘 만들어졌는지 확인하려면 데이터 사전을 찾아봐야 한다.

```
SELECT * FROM user_source
WHERE name='SP_SALARY';
```

- 프로시저의 삭제

  DROP명령으로 프로시저를 삭제할 수 있다.

```
DROP PROCEDURE sp_salary;
```

프로시저

- 프로시저의 데이터 입출력
  - 프로시저에서 매개변수(argument)는 단순히 프로시저가 실행될 때 전달받는 값이 아니다.
  - 값을 프로시저 내부로 전달 할 수도 반대로 프로시저 밖으로 전달 할 수도 있다.
    - IN 프로시저 내부에 값 전달
    - OUT 프로시저 밖으로 값 전달
    - INOUT 프로시저 안팎으로 값 전달

  - 프로시저 밖에서 OUT 매개변수의 값을 받기 위해서는
  외부에 선언된 변수가 필요하다
    - variable 변수명 타입(크기);
  - 바인드된 변수를 출력 : PRINT 변수명;

프로시저

- 예제 : 특정 사원의 이름으로 급여를 조회하기 해 보자

IN

```
CREATE PROCEDURE sp_salary_ename(
v_ename IN employee.ename%type)
IS
    v_salary employee.salary%type;
BEGIN
    SELECT salary INTO v_salary
    FROM employee
    WHERE ename=v_ename;
    dbms_output.put_line(v_ename||'의 급여는'||v_salary);
END;

EXECUTE sp_salary_ename('SCOTT');
EXECUTE sp_salary_ename('KING');
```

OUT

```
CREATE PROCEDURE sp_salary_ename2(
v_ename IN employee.ename%type,
v_salary OUT employee.salary%type)
IS
BEGIN
    SELECT salary INTO v_salary
    FROM employee
    WHERE ename=v_ename;
END;

variable v_salary varchar2(14);
EXECUTE sp_salary_ename2('SCOTT',:v_salary);
print v_salary;
```

함수

- 함수는 프로시저와 다르게 매개변수를 활용하지 않고 어떤 결과를 반환하기 위해 사용된다.

- 프로시저는 매개변수의 개수만큼 반환한다면 함수는 단 하나의 결과만을 반환한다.

함수

- 함수의 생성방법 : CREATE [OR REPLACE]

  CREATE [OR REPLACE] FUNCTION 함수명(매개변수)
  반환타입
  IS
  　　선언부
  BEGIN
  　　실행부
  END;

- 함수 실행 방법

함수

- 함수 사용 예

```
CREATE OR REPLACE FUNCTION fn_salary_ename(
v_ename IN employee.ename%type)
    RETURN number
IS
    v_salary number(7,2);
BEGIN
    SELECT salary INTO v_salary
    FROM employee
    WHERE ename = v_ename;
    RETURN v_salary;
END;
```

```
variable v_salary2 number;
EXECUTE :v_salary2:=fn_salary_ename('SCOTT');
print v_salary;
```

함수

- 함수 사용 예 - SELECT문 내부에서 함수를 사용 가능하다.
  사실상 내부 함수에 반대되는 사용자 정의 함수이다.

```
SELECT ename,fn_salary_ename('SCOTT') AS "급여"
FROM employee
WHERE ename='SCOTT';
```

트리거의 개념

- 트리거는 프로시저와 유사하지만 실행되는 원리가 다르다
- 프로시저는 작성 후 EXECUTE 명령어를 통해서 직접 실행하지만 트리거는 어떤 이벤트(주로 DML)가 발생했을 때 '내부적'으로 실행되는 저장된 프로시저이다.

트리거의 개념

- 트리거의 특징 : 프로시저의 형태이지만 단독으로 사용되기 보다 DML 구문에 붙여서 사용이 된다.

- 트리거의 종류
  - AFTER 트리거 : DML등의 작업이 일어났을 때 작동하는 트리거, 해당 작업 후에 작동한다.(테이블에만 작동)
  - BEFORE 트리거 : DML등의 작업이 일어났을 때 작동하는 트리거, 해당 작업 직전에 작동하다.

트리거의 개념

- 트리거를 사용하기에 앞서 실습용 테이블을 구성해봅니다.

```
CREATE TABLE dept_orgin
AS
SELECT * FROM department WHERE 0=1;

DROP TABLE dept_copy;

CREATE TABLE dept_copy
AS
SELECT * FROM department WHERE 0=1;
```

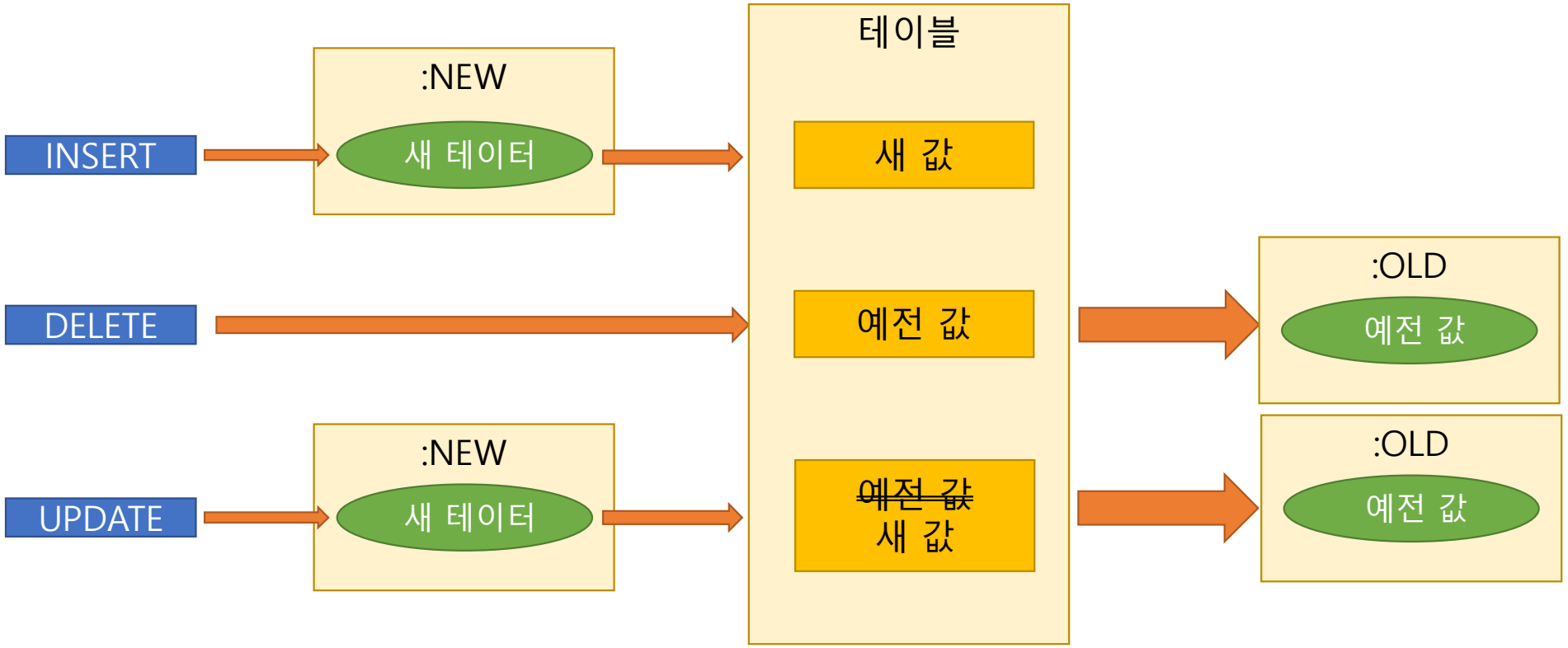
트리거의 개념

- INSERT에 적용할 트리거를 생성하기

```
CREATE OR REPLACE TRIGGER trigger_sample1
    AFTER INSERT
    ON dept_orgin
    for each row
BEGIN
    IF inserting then
        dbms_output.put_line('Insert Trigger 발생');
        insert into dept_copy
            values(:new.dno, :new.dname, :new.loc);
    END IF;
END;
```

트리거의 개념

- 트리거에서 생성하는 임시 테이블 :NEW, :OLD



트리거의 개념

- INSERT를 통해서 데이터를 추가해 봅니다.

```
INSERT INTO dept_orgin
VALUES(10,'ACCOUNTING','NEW YORK');
```

- origin테이블과 copy테이블 둘 다 데이터가 입력된 것을 확인 할 수 있다

트리거의 개념

- UPDATE와 DELETE 트리거를 만들어 보자
  - 백업 테이블에 컬럼을 추가한다.

```
ALTER TABLE dept_copy
ADD modType NCHAR(2);
```

  - 트리거를 만든다.

```
CREATE OR REPLACE TRIGGER trigger_sample2
    AFTER UPDATE OR DELETE
    ON dept_orgin
    for each row
DECLARE
    v_modType NCHAR(2);
BEGIN
    IF updating then
        dbms_output.put_line('Updating Trigger 발생');
        v_modType :='수정';
    ELSIF deleting then
        dbms_output.put_line('Deleting Trigger 발생');
        v_modType :='삭제';
    END IF;
    INSERT INTO dept_copy
    VALUES(:old.dno, :old.dname, :old.loc,v_modType);
END;
```

트리거의 개념

- UPDATE를 통해 데이터를 변경하고 copy테이블을 확인해 봅니다.

```
UPDATE dept_orgin SET dno=30, dname='SALES', loc='CHICAGO'
WHERE dno=10;

SELECT * FROM dept_orgin;
SELECT * FROM dept_copy;
```

- DELETE를 통해 데이터를 삭제하고 copy 테이블을 확인해 봅니다.

```
DELETE FROM dept_orgin
WHERE dno=30;

SELECT * FROM dept_orgin;
SELECT * FROM dept_copy;
```