저장 서브프로그램(stored subprogram)

PL/SQL 블록은 작성한 내용을 한 번 실행하는 데 사용한다.
PL/SQL 블록은 이름이 정해져 있지 않아서 익명 블록이라고 한다.
익명 블록은 오라클에 저장되지 않기 때문에 한 번 실행한 후에 다시 실행하려면 PL/SQL 블록을 다시 작성하여 실행해야 한다.

- 여러 번 사용할 목적으로 이름을 지정하여 **오라클에 저장**해 두는 PL/SQL 프로그램
- 저장할 때 한 번 컴파일 한다.
- 오라클에 저장하여 공유할 수 있어서 메모리, 성능, 재사용성 등 여러 면에서 장점이 있다.
- 대표적인 구현 방식은 프로시저, 함수, 패키지, 트리거이다

1. 프로시저

- 특정 처리 작업을 수행하는데 사용하는 저장 서브프로그램
- 파라미터를 사용할 수도 있고 사용하지 않을 수도 있다.

1) 파라미터를 사용하지 않는 프로시저

작업 수행에 별다른 입력 데이터가 필요하지 않은 경우

① 프로시저 생성

- 선언부, 실행부, 예외 처리부로 구성되어 있다

[형식]

```
CREATE [OR REPLACE] PROCEDURE 프로시저명
IS | AS          -- DECLARE 키워드는 사용하지 않는다.
    선언부
BEGIN
    실행부
EXCEPTION
    예외 처리부
END [프로시저 이름];
```

② 프로시저 실행

[형식]

```
EXECUTE 프로시저 이름;
```

③ 프로시저 내용 확인

④ 프로시저 삭제

[실습]

```
SET SERVEROUTPUT ON;

-- 프로시저 생성
CREATE OR REPLACE PROCEDURE PRO_NOPARAM
IS
    NAME VARCHAR2(10);
    AGE NUMBER(4) := 25;
BEGIN
    NAME := '홍길동';
    DBMS_OUTPUT.PUT_LINE('NAME : ' || NAME);
    DBMS_OUTPUT.PUT_LINE('AGE : ' || AGE);
END PRO_NOPARAM;
/

-- 프로시저 실행
EXECUTE PRO_NOPARAM;

-- PL/SQL에서 프로시저 실행
BEGIN
    PRO_NOPARAM;
END;
/

-- 프로시저 확인
SELECT * FROM USER_SOURCE WHERE NAME = 'PRO_NOPARAM';

-- 프로시저 삭제
DROP PROCEDURE PRO_NOPARAM;
```

2) 파라미터를 사용하는 프로시저

[형식]

```
CREATE [OR REPLACE] PROCEDURE 프로시저명
(
    파라미터 이름  [MODES] 자료형 [ := | DEFAULT ], -- 자리수 지정 및 NOT NULL 사용 불가능
    파라미터 이름  [MODES] 자료형 [ := | DEFAULT ], -- ;이 아니라 ,로 지정
    ...
)
IS | AS      -- DECLARE 키워드는 사용하지 않는다.
    선언부
BEGIN
    실행부
EXCEPTION
    예외 처리부
END [프로시저 이름];
```

① IN 모드 파라미터

- 프로시저 실행에 필요한 값을 직접 입력받는 형식의 파라미터를 지정할 때 사용한다.

[실습]

```
CREATE OR REPLACE PROCEDURE PRO_PARAM_IN
(
    NAME IN VARCHAR2,
    AGE NUMBER,
    PHONE VARCHAR2 := '010-1234-5678',
    ADDR VARCHAR2 DEFAULT '서울'
)
IS

BEGIN
    DBMS_OUTPUT.PUT_LINE('NAME : ' || NAME);
    DBMS_OUTPUT.PUT_LINE('AGE : ' || AGE);
    DBMS_OUTPUT.PUT_LINE('PHONE : ' || PHONE);
    DBMS_OUTPUT.PUT_LINE('ADDR : ' || ADDR);
END PRO_PARAM_IN;
/

EXECUTE PRO_PARAM_IN('홍길동', 25, '010-1111-1111', '부산');

EXECUTE PRO_PARAM_IN('홍길동', 25);

EXECUTE PRO_PARAM_IN(NAME => '홍길동', AGE => 25); -- 파라미터 이름과 값을 직접 대입
```

② OUT 모드 파라미터

- 프로시저 실행 후 호출한 프로그램으로 값을 반환한다.

[실습]

```
CREATE OR REPLACE PROCEDURE PRO_PARAM_OUT
(
    EMPNO IN EMPLOYEES.EMPLOYEE_ID%TYPE,
    EMPNAME OUT EMPLOYEES.LAST_NAME%TYPE,
    SAL OUT EMPLOYEES.SALARY%TYPE
)
IS

BEGIN
    SELECT LAST_NAME, SALARY INTO EMPNAME, SAL FROM EMPLOYEES WHERE EMPLOYEE_ID = EMPNO;
END PRO_PARAM_OUT;
/


DECLARE
    V_EMPNAME EMPLOYEES.LAST_NAME%TYPE;
    V_SAL EMPLOYEES.SALARY%TYPE;
BEGIN
    PRO_PARAM_OUT(100, V_EMPNAME, V_SAL);
    DBMS_OUTPUT.PUT_LINE('V_EMPNAME : ' || V_EMPNAME);
    DBMS_OUTPUT.PUT_LINE('V_SAL : ' || V_SAL);
END;
/
```

③ IN OUT 모드 파라미터

- IN, OUT으로 선언한 파라미터 기능을 동시에 수행한다.
- 입력받을 때와 프로시저 수행 후 결과 값을 반환할 때 사용한다.

[실습]

```
CREATE OR REPLACE PROCEDURE PRO_PARAM_INOUT
(
    NUM IN OUT NUMBER
)
IS

BEGIN
    NUM := NUM * 2;
END PRO_PARAM_INOUT;
/

DECLARE
    NUM NUMBER;
BEGIN
    NUM := 5;
    PRO_PARAM_INOUT(NUM);
    DBMS_OUTPUT.PUT_LINE('NUM : ' || NUM);
END;
/
```

2. 함수

- 오라클 함수는 내장함수와 사용자 정의 함수로 나눌 수 있다.
- 함수는 반환 값의 자료형과 실행부에서 반환할 값을 RETURN절 및 RETURN문으로 명시해야 한다.

함수 생성

[형식]

```
CREATE [OR REPLACE] FUNCTION 함수명
(
    파라미터 이름  [IN] 자료형 [ := | DEFAULT ], -- 프로시저와 달리 IN모드만 지정한다.
    파라미터 이름  [IN] 자료형 [ := | DEFAULT ], -- ;이 아니라 ,로 지정
    ...
)
RETURN 자료형
IS | AS
    선언부
BEGIN
    실행부
    RETURN (반환값);
EXCEPTION
    예외 처리부
END [함수 이름];
```

[실습]

```
-- 함수 생성
CREATE OR REPLACE FUNCTION FUNC_TAX
(
    SALARY IN NUMBER
)
RETURN NUMBER
IS
    TAX NUMBER := 0.033;
BEGIN
    RETURN (ROUND(SALARY - (SALARY * TAX)));
END FUNC_TAX;
/

-- 함수 실행
DECLARE
    PAY NUMBER;
BEGIN
    PAY := FUNC_TAX(3000000);
    DBMS_OUTPUT.PUT_LINE('PAY : ' || PAY);
END;
/
```

```
-- SQL문에서 함수 실행
SELECT FUNC_TAX(5000000) FROM DUAL;

-- 함수 삭제
DROP FUNCTION FUNC_TAX;
```

3. 트리거 (Trigger)

- 데이터베이스 안의 특정 상황이나 동작, 즉 이벤트가 발생하면 자동으로 실행되는 기능을 정의하는 PL/SQL 서브프로그램이다.
- 테이블의 데이터를 특정 사용자가 변경하려 할 때 해당 데이터나 사용자 기록을 확인한다든지 상황에 따라 데이터를 변경하지 못하게 막는 것이 가능하다.
- 데이터베이스가 가동되거나 종료될 때 데이터베이스 관리자에게 메일을 보내는 기능도 구현할 수 있다.
- 트리거는 특정 이벤트가 발생할 때 자동으로 작동하는 서브프로그램이므로,
  프로시저, 함수와 같이 EXECUTE 또는 PL/SQL 블록에서 따로 실행하지 못한다.

DML 트리거

[형식]

```
CREATE [OR REPLACE] TRIGGER 트리거 이름
BEFORE | AFTER
INSERT | UPDATE | DELETE ON 테이블명
REFERENCING OLD as old | New as new
FOR EACH ROW WHEN 조건식
FOLLOW 트리거 이름, 트리거 이름,...
ENABLE | DISABLE

DECLARE
    선언부
BEGIN
    실행부
EXCEPTION
    예외 처리
END;
```

[실습]

① BEFORE

- DML 명령어가 실행하기 전에 작동하는 트리거가 생성

```
CREATE TABLE EMP_TAB AS SELECT * FROM EMPLOYEES;
SELECT * FROM EMP_TAB;

CREATE OR REPLACE TRIGGER TRI_EMP_WEEKEND
-- EMP_TAB 테이블에 INSERT, UPDATE, DELETE 실행 될 때 트리거 작동
BEFORE
INSERT OR UPDATE OR DELETE ON EMP_TAB
BEGIN
    IF TO_CHAR(SYSDATE, 'DY') IN ('토', '일') THEN
        IF INSERTING THEN
            -- RAISE_APPLICATION_ERROR 프로시저를 사용하여 사용자 정의 예외를 발생
            -- RAISE_APPLICATION_ERROR(오류코드, 에러메시지);
```

```
            -- 오류코드: -20000 ~ -20999
            RAISE_APPLICATION_ERROR(-20000, '주말 사원정보 추가 불가');
        ELSIF UPDATING THEN
            RAISE_APPLICATION_ERROR(-20001, '주말 사원정보 수정 불가');
        ELSIF DELETING THEN
            RAISE_APPLICATION_ERROR(-20002, '주말 사원정보 삭제 불가');
        ELSE
            RAISE_APPLICATION_ERROR(-20003, '주말 사원정보 변경 불가');
        END IF;
    END IF;
END;
/

-- 평일에는 수정이 잘 된다.
UPDATE EMP_TAB SET SALARY=30000 WHERE EMPLOYEE_ID = 100;
SELECT * FROM EMP_TAB;

-- 시스템의 날짜를 주말로 변경
UPDATE EMP_TAB SET SALARY=24000 WHERE EMPLOYEE_ID = 100

오류 보고 -
ORA-20001: 주말 사원정보 수정 불가
ORA-06512: "HR.TRI_EMP_WEEKEND",  6행
ORA-04088: 트리거 'HR.TRI_EMP_WEEKEND'의 수행시 오류

-- INSERT 에러 코드 확인

-- DELETE 에러 코드 확인
```

② AFTER
- DML 명령어가 실행된 후에 작동하는 트리거가 생성
- EMP_TAB에 DML 명령어가 실행되면 EMP_TAB_LOG에 행이 추가된다.

```
-- 테이블 생성
CREATE TABLE EMP_TAB_LOG(
    TABLENAME VARCHAR2(10), -- 테이블 이름
    DML_TYPE VARCHAR2(10),  -- DML 명령어
    EMPNO NUMBER(4),        -- 사원번호
    EMPNAME VARCHAR2(30),   -- USER(계정) 이름
    CHANGE_DATE DATE);

SELECT * FROM EMP_TAB_LOG;

-- 트리거 생성
CREATE OR REPLACE TRIGGER TRI_EMP_LOG
-- EMP_TAB 테이블에 INSERT, UPDATE, DELETE 실행 후에 트리거 작동
AFTER
INSERT OR UPDATE OR DELETE ON EMP_TAB
FOR EACH ROW -- 행 별로 트리거 작동

BEGIN
    IF INSERTING THEN
        INSERT INTO EMP_TAB_LOG
        VALUES ('EMP_TAB',
                'INSERT',
                :NEW.EMPLOYEE_ID, --새로 추가 EMPLOYEE_ID
                SYS_CONTEXT('USERENV', 'SESSION_USER'), -- 현재 데이터베이스에 접속한 사용자
                SYSDATE);
    ELSIF UPDATING THEN
        INSERT INTO EMP_TAB_LOG
        VALUES ('EMP_TAB',
                'UPDATE',
                :OLD.EMPLOYEE_ID, -- 변경 전 EMPLOYEE_ID
                SYS_CONTEXT('USERENV', 'SESSION_USER'),
                SYSDATE);
    ELSIF DELETING THEN
        INSERT INTO EMP_TAB_LOG
        VALUES ('EMP_TAB',
                'DELETE',
                :OLD.EMPLOYEE_ID, -- 변경 전 EMPLOYEE_ID
                SYS_CONTEXT('USERENV', 'SESSION_USER'),
                SYSDATE);
    END IF;
END;
```

```
/

-- 평일에는 추가가 잘 된다.
-- INSERT 할 때 EMAIL은 NOT NULL 이므로 반드시 입력해야 한다.
INSERT INTO EMP_TAB(EMPLOYEE_ID, LAST_NAME, EMAIL, JOB_ID, HIRE_DATE)
VALUES (9999, 'Test', 'Test', 'Test', SYSDATE);
SELECT * FROM EMP_TAB;
SELECT * FROM EMP_TAB_LOG; -- 1개 로그가 생겼다

-- EMP_TAB 테이브은 EMPLOYEES 테이블 복제한 것이라서, 제약조건이 NOT NULL만 존재
-- EMPLOYEE_ID 에 똑같은 9999를 추가해도 입력이 된다.
INSERT INTO EMP_TAB(EMPLOYEE_ID, LAST_NAME, EMAIL, JOB_ID, HIRE_DATE)
VALUES (9999, 'Test', 'Test', 'Test', SYSDATE);
SELECT * FROM EMP_TAB;
SELECT * FROM EMP_TAB_LOG; -- 1개 로그가 생겼다

-- 평일에는 수정이 잘된다.
UPDATE EMP_TAB SET SALARY=35000 WHERE MANAGER_ID = 100; -- 14개 수정
SELECT * FROM EMP_TAB;
SELECT * FROM EMP_TAB_LOG; -- 14개 로그가 생겼다.

-- 트리거 관리
SELECT * FROM USER_TRIGGERS;

-- 트리거 상태(활성 / 비활성) 변경
ALTER TRIGGER TRI_EMP_LOG DISABLE;
UPDATE EMP_TAB SET SALARY=50000 WHERE EMPLOYEE_ID = 9999;
SELECT * FROM EMP_TAB;
SELECT * FROM EMP_TAB_LOG; -- 로그가 생성되지 않았다

ALTER TRIGGER TRI_EMP_LOG ENABLE;

-- 트리거 삭제
DROP TRIGGER TRI_EMP_LOG;
```