ch19. 저장 서브프로그램

19-1. 저장 서브프로그램

> 저장 서브프로그램이란?

- 익명 블록 *anonymous block* : 이름이 정해져 있지 않은 PL/SQL블록
- 익명 블록은 오라클에 저장되지 않기 때문에 한 번 실행한 뒤에 다시 실행 하려면 다시 작성하여 실행해야 한다.
- 그런데, PL/SQL로 만든 프로그램을 주기적으로 또는 필요할 때마다 여러 번 사용해야 하는 상황이 빈번히 발생.
    - 이럴 경우, 프로그램을 오라클에 저장해 두고 실행 가능하다.
- 저장 서브프로그램 *stored subprogram* : 여러 번 사용할 목적으로 이름을 지정하여 오라클에

  저장해 두는 PL/SQL 프로그램
- 오라클에서의 저장 서브프로그램 구현 방식
    - 프로시저, 함수, 패키지, 트리거
        - 저장 프로시저 : 일반적으로 특정 처리 작업 수행을 위한 서브 프로그램으로

          SQL문에서는 사용할수 없다.
        - 저장 함수 : 일반적으로 특정 연산을 거친 결과 값을 반환하는 서브프로그램으로

          SQL문에서 사용할 수 있다.
        - 패키지 : 저장 서브프로그램을 그룹화하는 데 사용한다.
        - 트리거 : 특정 상황 (이벤트) 이 발생할 때 자동으로 연달아 수행할 기능을 구현하는 데

          사용한다.

19-2. 프로시저

저장 프로시저는 특정 처리 작업을 수행하는 데 사용하는 저장 서브프로그램으로 용도에 따라 파라미터를 사용할 수 있고 사용하지 않을 수도 있다.

파라미터를 사용하지 않는 프로시저

프로시저 생성하기

- 작업 수행에 별다른 입력 데이터가 필요하지 않을 경우에 파라미터를 사용하지 않는 프로시저를 사용한다.

```
CREATE [OR REPLACE] PROCEDURE 프로시저 이름
IS | AS
   선언부
BEGIN
  실행부
EXCEPTION
  예외 처리부
END [프로시저 이름];
```

[OR REPLACE]	지정한 프로시저 이름을 가진 프로시저가 이미 존재하는 경우에 현재 작성한 내용으로 대체. 즉 덮어쓴다는 뜻이며 생략 가능한 옵션이다.
프로시저 이름	저장할 프로시저의 고유 이름을 지정. 같은 스키마 내에서 중복될 수 없다.
IS | AS	선언부를 시작하기 위해 IS 또는 AS 키워드를 사용. 선언부가 존재하지 않더라도 반드시 명시. DECLARE 키워드는 사용하지 않는다.
EXCEPTION	예외 처리부는 생략 가능.
END	프로시저 생성의 종료를 뜻하며 프로시저 이름은 생략 가능.

- 프로시저 생성하기

```
CREATE OR REPLACE PROCEDURE pro_noparam
IS
   V_EMPNO NUMBER(4) := 7788;
   V_ENAME VARCHAR2(10);
BEGIN
   V_ENAME := 'SCOTT';
   DBMS_OUTPUT.PUT_LINE('V_EMPNO : ' || V_EMPNO);
   DBMS_OUTPUT.PUT_LINE('V_ENAME : ' || V_ENAME);
END;
/
```

SQL*PLUS로 프로시저 실행하기

```
EXECUTE 프로시저 이름;
```

- 생성한 프로시저 실행하기

```
$ SET SERVEROUTPUT ON;
$ EXECUTE pro_noparam;
```

PL/SQL 블록에서 프로시저 실행하기

```
BEGIN
  프로시저 이름;
END;
```

- 익명 블록에서 프로시저 실행하기

```
BEGIN
   pro_noparam;
END;
/
```

프로시저 내용 확인하기

- 이미 저장되어 있는 프로시저를 포함하여 서브프로그램의 소스 코드 내용을 확인하려면 USER_SOURCE 데이터 사전에서 조회한다.

USER_SOURCE 의 열	설명
NAME	서브 프로그램(생성 객체) 이름
TYPE	서브 프로그램 종류 (PROCEDURE, FUNCTION 등)
LINE	서브프로그램에 작성한 줄 번호
TEXT	서브프로그램에 작성한 소스 코드

- USER_SOURCE를 통해 프로시저 확인하기(오라클)

```
SELECT *
  FROM USER_SOURCE
 WHERE NAME = 'PRO_NOPARAM';
```

- USER_SOURCE를 통해 프로시저 확인하기(SQL*PLUS)

```
$ SELECT TEXT
    FROM USER_SOURCE
   WHERE NAME = 'PRO_NOPARAM';
```

프로시저 삭제하기

```
$ DROP PROCEDURE PRO_NOPARAM;
```

파라미터를 사용하는 프로시저

- 프로시저를 실행하기 위해 입력 데이터가 필요한 경우에 파라미터를 정의할 수 있다.
  - 파라미터는 여러 개 작성할 수 있다.

```
CREATE [OR REPLACE] PROCEDURE 프로시저 이름

[(파라미터 이름1 [modes] 자료형 [ := | DEFAULT 기본값],
  파라미터 이름2 [modes] 자료형 [ := | DEFAULT 기본값],
   ...
  파라미터 이름2 [modes] 자료형 [ := | DEFAULT 기본값]
)]
IS | AS
   선언부
BEGIN
  실행부
EXCEPTION
  예외 처리부
END [프로시저 이름];
```

- 파라미터를 지정할 때 사용하는 모드

IN	지정하지 않으면 기본값으로 프로시저를 호출할 때 값을 입력받는다.
OUT	호출할 때 값을 반환한다.
IN OUT	호출할 때 값을 입력받은 후 실행 결과 값을 반환한다.

IN 모드 파라미터

- 프로시저에 파라미터 지정하기

```
CREATE OR REPLACE PROCEDURE pro_param_in
(
   param1 IN NUMBER,
   param2 NUMBER,
   param3 NUMBER := 3,
   param4 NUMBER DEFAULT 4
)
IS

BEGIN
   DBMS_OUTPUT.PUT_LINE('param1 : ' || param1);
   DBMS_OUTPUT.PUT_LINE('param2 : ' || param2);
   DBMS_OUTPUT.PUT_LINE('param3 : ' || param3);
   DBMS_OUTPUT.PUT_LINE('param4 : ' || param4);
END;
/
```

- 파라미터를 입력하여 프로시저 사용하기

```
EXECUTE pro_param_in(1,2,9,8);
```

- 기본값이 지정된 파라미터 입력을 제외하고 프로시저 사용하기

```
EXECUTE pro_param_in(1, 2);
```

- 실행에 필요한 개수보다 적은 파라미터를 입력하여 프로시저 실행하기

```
EXECUTE pro_param_in(1);
```

- 파라미터 이름을 활용하여 프로시저에 값 입력하기

```
EXECUTE pro_param_in(param1 => 10, param2 => 20);
```

- 파라미터 값을 지정할 때는 다음 세 가지 지정 방식을 사용할 수 있다.

종류	설명
위치 지정	지정한 파라미터 순서대로 값을 지정하는 방식
이름 지정	=> 연산자로 파라미터 이름을 명시하여 값을 지정하는 방식
혼합 지정	일부 파라미터는 순서대로 값만 지정하고 일부 파라미터는 => 연산자로 값을 지정하는 방식

OUT 모드 파라미터

OUT 모드를 사용한 파라미터는 프로시저 실행 후 호출한 프로그램으로 값을 반환한다.

- OUT 모드 파라미터 정의하기

```
CREATE OR REPLACE PROCEDURE pro_param_out
(
   in_empno IN EMP.EMPNO%TYPE,
   out_ename OUT EMP.ENAME%TYPE,
   out_sal OUT EMP.SAL%TYPE
)
IS

BEGIN
   SELECT ENAME, SAL INTO out_ename, out_sal
     FROM EMP
    WHERE EMPNO = in_empno;
END pro_param_out;
/
```

- OUT 모드 파라미터 사용하기

```
DECLARE
   v_ename EMP.ENAME%TYPE;
   v_sal EMP.SAL%TYPE;
BEGIN
   pro_param_out(7788, v_ename, v_sal);
   DBMS_OUTPUT.PUT_LINE('ENAME : ' || v_ename);
   DBMS_OUTPUT.PUT_LINE('SAL : ' || v_sal);
END;
/
```

IN OUT 모드 파라미터

IN OUT 모드로 선언한 파라미터는 IN, OUT 으로 선언한 파라미터 기능을 동시에 수행한다 . 즉 , 값을 입력받을 때와 프로시저 수행 후 결과값을 반환 할 때 사용한다.

- IN OUT 모드 파라미터 정의하기

```
CREATE OR REPLACE PROCEDURE pro_param_inout
(
   inout_no IN OUT NUMBER
)
IS

BEGIN
   inout_no := inout_no * 2;
END pro_param_inout;
/
```

- IN OUT 모드 파라미터 사용하기

```
DECLARE
   no NUMBER;
BEGIN
   no := 5;
   pro_param_inout(no);
   DBMS_OUTPUT.PUT_LINE('no : ' || no);
END;
/
```

프로시저 오류 정보 확인하기

- 다음 방법은 다른 서브프로그램의 오류에도 똑같이 적용할 수 있다.

- 생성할 때 오류가 발생하는 프로시저 알아보기

```
CREATE OR REPLACE PROCEDURE pro_err
IS
   err_no NUMBER;
BEGIN
   err_no = 100;
   DBMS_OUTPUT.PUT_LINE('err_no : ' || err_no);
```

```
END pro_err;
/
```

⇒ 서브프로그램을 만들 때 발생한 요류는 SHOW ERRORS명령어와 USER_ERRORS 데이터 사전을
조회하여 확인할 수 있다.

SHOW ERRORS로 오류 확인

```
SHOW ERRORS;
```

- 만약 최근에 발생한 프로그램 오류가 아니라 특정 프로그램의 오류 정보를 확인하고 싶다면
  프로그램 종류와 이름을 추가로 지정하면 된다.

  ```
  SHOW ERR 프로그램종류 프로그램이름;
  SHOW ERR PROCEDURE pro_err;
  ```

USER_ERRORS로 오류 확인하기

```
SELECT *
  FROM USER_ERRORS
 WHERE NAME = 'PRO_ERR';
```