ORACLE

19.프로시저&함수&커서

목차

1. 프로시저(1)

- 지금까지 실습한 예제는 한번 실행하면 결과값을 돌려주고 끝나는 예제였다.

- 경우에 따라서는 우리가 만든 PL/SQL을 저장해 놓고 필요한 경우 호출하여 사용할 수 있었으면 할 때가 있다. 오라클은 사용자가 만든 PL/SQL 문을 데이터베이스에 저장 할 수 있도록 저장 프로시저라는 것을 제공한다.

- 이렇게 저장 프로시저를 사용하면 복잡한 DML 문들을 필요할 때마다 다시 입력할 필요 없이 간단하게 호출만 해서 복잡한 DML 문의 실행 결과를 얻을 수 있다.

- 저장 프로시저를 사용하면 성능도 향상되고, 호환성 문제도 해결된다.

1. 프로시저(2)

- 저장 프로시저를 생성하기 위한 CREATE PROCEDURE의 형식은 다음과 같다.

형식

```
CREATE [OR REPLACE ] PROCEDURE prcedure_name
( argument1 [mode] data_taye,
  argument2 [mode] data_taye . . .
)
IS
local_variable declaration

BEGIN
statement1;
statement2;
. . .
END;
/
```

1. 프로시저(3)

- 저장 프로시저를 생성하려면 CREATE PROCEDURE 다음에 새롭게 생성하고자 하는 프로시저 이름을 기술한다.

- 이렇게 해서 생성한 저장 프로시저는 여러 번 반복해서 호출해서 사용할 수 있다는 장점이 있다.

- 생성된 저장 프로시저를 제거하기 위해서는 DROP PROCEDURE 다음에 제거하고자 하는 프로시저 이름을 기술한다.

- OR REPLACE 옵션은 이미 학습한 대로 이미 같은 이름으로 저장 프로시저를 생성할 경우 기존 프로시저는 삭제하고 지금 새롭게 기술한 내용으로 재 생성하도록 하는 옵션이다.

1. 프로시저(4)

- 프로시저는 어떤 값을 전달받아서 그 값에 의해서 서로 다른 결과물을 구하게 된다.
- 값을 프로시저에 전달하기 위해서 프로시저 이름 다음에 괄호로 둘러 싼 부분에 전달 받을 값을 저장할 변수를 기술한다.
- 이를 ARGUMENT 우리나라 말로 매개 변수라 한다.
- 프로시저는 매개 변수의 값에 따라 서로 다른 동작을 수행하게 된다.
- [MODE] 는 IN과 OUT, INOUT 세 가지를 기술할 수 있는데 IN은 데이터를 전달 받을 때 쓰고 OUT은 수행된 결과를 받아갈 때 사용한다. INOUT은 두 가지 목적에 모두 사용된다.

1.1 프로시저 생성하기

- ex) 사원 테이블에 저장된 모든 사원을 삭제하는 프로시저를 작성해보도록 하겠다.
- 1. 모든 사원을 삭제하는 프로시저를 실행시키기 위해서 미리 사원 테이블을 복사해서 새로운 사원 테이블(EMP01)을 만들어 놓는다.
- 2. 다음과 같이 입력하시오.

예

```
CREATE OR REPLACE PROCEDURE DEL_ALL
IS
BEGIN
DELETE FROM EMP01;
END;
/
```

- 3. 생성된 저장 프로시저는 EXECUTE 명령어로 실행시킨다.

예

```
EXECUTE DEL_ALL
```

1.2 프로시저 오류 원인 살피기

- "프로시저가 생성되었습니다."란 메시지와 함께 한 번에 저장 프로시저를 성공적으로 생성할 수도 있지만, "경고: 컴파일 오류와 함께 프로시저가 생성되었습니다" 와 같은 메시지가 출력되면 저장 프로시저를 생성하는 과정에서 오류가 발생해서 저장 프로시저 생성에 실패하는 경우이다.
- 이럴 경우에는 오류를 제거해서 다시 저장프로서지를 생성해야 하는데, 오류 메시지를 모른 채 오류를 수정하기란 쉽지 않다.
- 오류가 발생할 경우 "SHOW ERROR" 명령어를 수행하면 오류가 발생한 원인을 알 수 있게 된다.
- 원인을 분석하여 오류를 수정한 후 다시 저장 프로시저를 생성을 시도하여 '프로시저가 생성되었습니다.'란 메시지가 출력되어 저장 프로시저가 성공적으로 생성될 때까지 오류 수정 작업을 반복해야 한다.

1.2.1 프로시저 오류 처리하기(1)

- 저장 프로시저를 작성시 오류가 발생하면 이를 수정하기 위한 방법을 살펴보자.
- 1. 다음과 같이 입력하시오.

예

```
CREATE OR REPLACE PROCEDURE DEL_ALL
IS
DELETE FROM EMP01;
END;
/
```

- 2. 해당 내용을 실행하면 '경고: 컴파일 오류와 함께 프로시저가 생성되었습니다' 혹은 에러를 발생시킨다.
- 3. 발생한 오류에 대한 정보를 알고 싶을 때에는 SHOW ERROR를 입력한다.

예

```
SHOW ERROR;
```

1.2.2 프로시저 오류 처리하기(2)

```
LINE/COL ERROR
-------- -----------------------------------------------------------------
3/8      PLS-00103: Encountered the symbol "FROM" when expecting one of the following:
         constant exception <an identifier>
         <a double-quoted delimited-identifier> table long double ref
         char time timestamp interval date binary national character
         nchar
```

- 4. 내용을 수정한 후 프로시저를 재 생성하여 실행시킨다.

2. 프로시저 조회하기

- 저장 프로시저를 작성한 후 사용자가 저장 프로시저가 생성되었는지 확인하려면 USER_SOURCE 살펴보면 된다.
- 다음은 USER_SOURCE의 구조다.

형식 **DESC USER_SOURCE**

- USER_SOURCE의 내용을 조회하면 어떤 저장 프로시저가 생성되어 있는지와 해당 프로시저의 내용이 무엇인지 확인할 수 있다.

3. 프로시저 매개 변수(1)

- DEL_ALL 저장 프로시저는 사원 테이블의 모든 내용을 삭제한다.
- 만일 특정 사원만을 삭제하려면 어떻게 해야할까
- 저장 프로시저를 생성할 때 삭제하고자 하는 사원의 이름이나 사원 번호를 프로시저에 전달해 주어 이와 일치하는 사원을 삭제하면 된다.
- 저장 프로시저에 값을 전달해 주기 위해서 매개 변수를 사용된다.

3. 프로시저 매개 변수(2)

- 매개변수가 있는 저장프로시저는 다음과 같이 정의한다.

형식

```
CREATE OR REPLACE PROCEDURE
DEL_ENAME(VENAME EMP01.ENAME%TYPE)
IS
BEGIN
DELETE FROM EMP01 WHERE ENAME=VENAME;
END;
/
```

- 저장 프로시저 이름인 DEL_ENAME 다음에 ( )를 추가하여 그 안에 선언한 변수가 매개 변수이다. 이 매개변수에 값은 프로시저를 호출할 때 전달해 준다.

형식

```
EXECUTE DEL_ENAME('김사랑');
```

4. IN, OUT, INOUT 매개 변수

- CREATE PROCEDURE로 프로시저를 생성할 때 MODE를 지정하여 매개변수를 선언할 수 있는데 MODE 에 IN, OUT, INOUT 세 가지를 기술할 수 있다.

- IN은 데이터를 전달 받을 때 쓰고 OUT은 수행된 결과를 받아갈 때 사용한다.

- INOUT은 두 가지 목적에 모두 사용된다.

- 이번에는 MODE를 지정하면 어떠한 기능이 부여되는지 자세히 살펴보기로하자.

4.1 IN 매개 변수

- 앞선 예제 중에서 매개변수로 사원의 이름을 전달받아서 해당 사원을 삭제하는 프로시저인 DEL_ENAME를 작성해보았다.
- DEL_ENAME 프로시저에서 사용된 매개변수는 프로시저를 호출할 때 기술한 값을 프로시저 내부에서 받아서 사용하고 있다.

```
EXECUTE DEL_ENAME('김사랑');
```

```
CREATE OR REPLACE PROCEDURE DEL_ENAME
(VENAME EMP01.ENAME%TYPE)
```

- 이렇게 프로시저 호출 시 넘겨준 값을 받아오기 위한 매개변수는 MODE를 IN으로 지정해서 선언한다.

```
CREATE PROCEDURE DEL_ENAME
(VENAME IN EMP01.ENAME%TYPE)
```

4.2 OUT 매개 변수(1)

- 프로시저에 구한 결과 값을 얻어 내기 위해서는 MODE를 OUT으로 지정한다.
- 다음은 저장 프로시저를 생성할 때 IN, OUT 매개변수를 사용한 예이다.

예

```
CREATE OR REPLACE PROCEDURE SEL_EMPNO
( VEMPNO IN EMP.EMPNO%TYPE,
  VENAME OUT EMP.ENAME%TYPE,
  VSAL OUT EMP.SAL%TYPE,
  VJOB OUT EMP.JOB%TYPE
)
IS
BEGIN
  SELECT ENAME, SAL, JOB
  INTO VENAME, VSAL, VJOB
  FROM EMP
  WHERE EMPNO=VEMPNO;
END;
/
```

4.2 OUT 매개 변수(2)

- 저장 프로시저 SEL_EMPNO를 호출할 때에도 이전과 마찬가지로 EXECUTE 명령어로 실행시킨다. OUT 매개변수에는 값을 받아오기 위해서는 프로시저 호출 시 변수 앞에 ':'를 덧붙인다.

예

```
EXECUTE
SEL_EMPNO(7788, :VAR_ENAME, :VAR_SAL, :VAR_JOB)
```

- :를 덧붙여주는 변수는 미리 다음과 같이 선언되어 있어야 합니다.

예

```
VARIABLE VAR_ENAME VARCHAR2(15);
VARIABLE VAR_SAL NUMBER;
VARIABLE VAR_JOB VARCHAR2(9);
```

- 위와 같이 선언한 변수를 "바인드 변수"라고 한다.

- Print명령으로 바인드 변수를 출력시키기

예

```
PRINT VAR_ENAME
PRINT VAR_SAL
PRINT VAR_JOB
```

5. 저장 함수 생성(1)

- 저장 함수는 저장 프로시저와 거의 유사한 용도로 사용한다.
- 차이점이라곤 함수는 실행 결과를 되돌려 받을 수 있다는 점이다.
- 다음은 저장 함수를 만드는 기본 형식이다.

예

```
CREATE [OR REPLACE ] FUNCTION function_name
( argument1 [mode] data_taye,
  argument2 [mode] data_taye . . .
)
RETURN data_type;
IS
BEGIN
statement1;
statement2;
RETURN variable_name;
END;
```

5. 저장 함수 생성(2)

- 프로시저를 만들 때에는 PROCEDURE라고 기술하지만, 함수를 만들 때에는 FUNCTION이라고 기술한다.
- 함수는 결과를 되돌려 받기 위해서 함수가 되돌려 받게 되는 자료 형과 되돌려 받을 값을 기술해야 한다.
- 저장 함수는 호출결과를 얻어오기 위해서 호출 방식에 있어서도 저장 프로시저와 차이점이 있다.

예 **EXECUTE :varible_name := function_name(argument_list);**

5.1 저장 함수 작성하기(1)

- ex) 다음은 특별 보너스를 지급하기 위한 저장 함수를 작성해 봅시다. 보너스는 급여의 200%를 지급한다.
- 1. 다음과 같이 입력하시오.

예

```
CREATE OR REPLACE FUNCTION CAL_BONUS(
    VEMPNO IN EMP.EMPNO%TYPE )
RETURN NUMBER
IS
    VSAL NUMBER(7, 2);
BEGIN
    SELECT SAL INTO VSAL
    FROM EMP
    WHERE EMPNO = VEMPNO;

    RETURN (VSAL * 2);
END;
/
```

5.1 저장 함수 작성하기(2)

- 2. 함수의 결과 값을 저장할 변수를 선언한 후에 아래와 같이 함수를 호출한다.

예

```
VARIABLE VAR_RES NUMBER;
EXECUTE :VAR_RES := CAL_BONUS(1001);
```

- 3. 출력

예

```
PRINT VAR_RES
```

6. 커서(1)

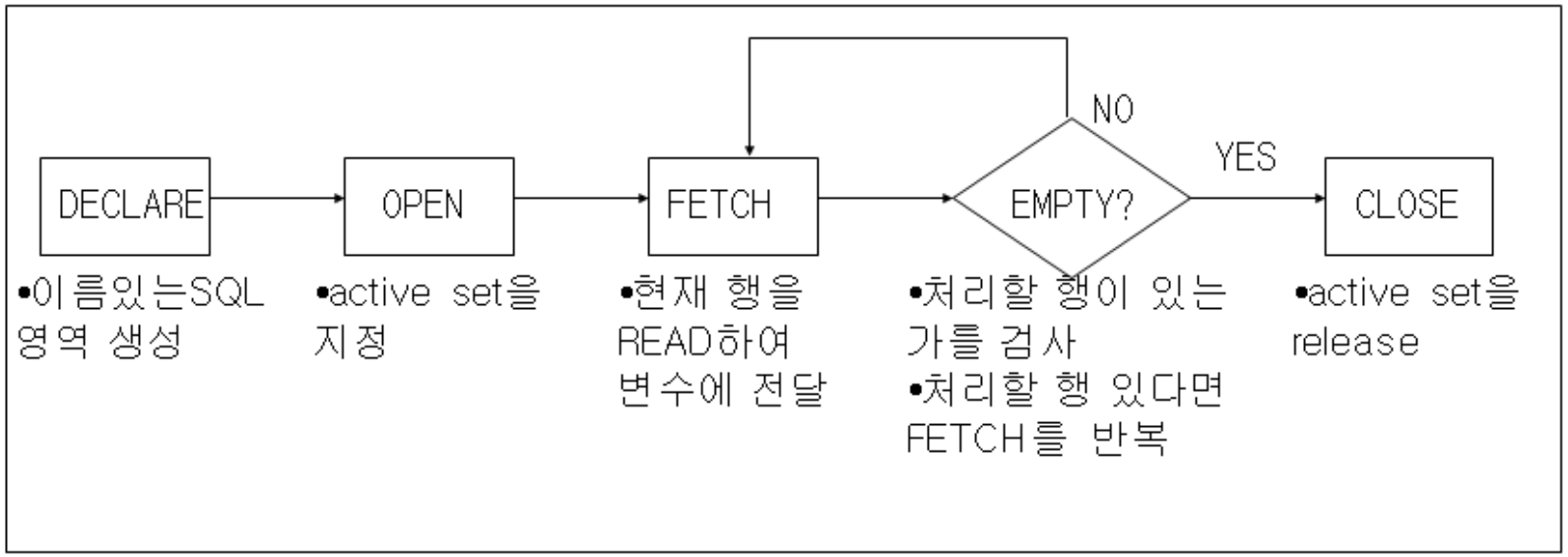
- 앞선 PL/SQL 예제에서는 처리 결과가 1개의 행인 SELECT문 만을 다루었다.
- 하지만 대부분의 SELECT 문은 수행 후 반환되는 행의 개수가 한 개 이상이다.
- 처리 결과가 여러 개의 행으로 구해지는 SELECT 문을 처리하려면 지금부터 학습할 커서를 이용해야 한다.

6. 커서(2)

형식

```
DECLARE
-- 커서 선언
CURSOR cursor_name IS statement;
BEGIN
-- 커서 열기
OPEN cursor_name;
--커서로부터 데이터를 읽어와 변수에 저장
FECTCH cur_name INTO variable_name;
--커서 닫기
CLOSE cursor_name;
END;
```

6. 커서(3)



6.1 DECLARE CURSOR

- 명시적으로 CURSOR를 선언하기 위해 CURSOR문장을 허용한다.

형식 **CURSOR *cursor_name* IS**
***select_statement*;**

구문	의미
cursor_name	PL/SQL 식별자
select_statement	INTO절이 없는 SELECT 문장

6.2 OPEN CURSOR

- 질의를 수행하고 검색 조건을 충족하는 모든 행으로 구성된 결과 셋을 생성하기 위해 CURSOR를 OPEN한다. CURSOR는 이제 결과 셋에서 첫 번째 행을 가리킨다.

형식 **OPEN *cursor_name*;**

- 부서 테이블의 모든 내용을 조회하는 **SELECT**문과 연결된 커서 **C1**을 오픈한다.

예 **OPEN C1;**

- **C1**을 오픈하면 검색 조건에 만족하는 모든 행으로 구성된 결과 셋이 구해지고 부서 테이블의 첫 번째 행을 가리키게 된다.

6.3 FETCH CURSOR(1)

- FETCH 문은 결과 셋에서 로우 단위로 데이터를 읽어 들인다. 각 인출 (FETCH) 후에 CURSOR는 결과 셋에서 다음 행으로 이동한다.

형식 **FETCH *cursor_name* INTO {*variable1[,variable2, ...]*};**

- FETCH 문장은 현재 행에 대한 정보를 얻어 와서 INTO 뒤에 기술한 변수에 저장한 후 다음 행으로 이동한다. 얻어진 여러 개의 로우에 대한 결과값을 모두 처리하려면 반복문에 FETCH 문을 기술해야 한다.

예

```
LOOP
FETCH C1 INTO VDEPT.DEPTNO, VDEPT.DNAME, VDEPT.LOC;
EXIT WHEN C1%NOTFOUND;
END LOOP;
```

6.3 FETCH CURSOR(2)

- 커서가 끝에 위치하게 되면 반복문을 탈출해야 한다..
- 단순 LOOP는 내부에 EXIT WHEN 문장을 포함하고 있다가 EXIT WHEN 다음에 기술한 조건에 만족하면 단순 LOOP을 탈출하게 된다.
- 반복문을 탈출할 조건으로 "C1%NOTFOUND"를 기술하였다.
- NOTFOUND는 커서의 상태를 알려주는 속성 중에 하나인데 커서 영역의 자료가 모두 FETCH됐다면 TRUE를 되돌린다.
- 커서 C1영역의 자료가 모두 FETCH 되면 반복문을 탈출하게 된다.

6.4 커서의 상태

- FETCH 문을 설명하면서 커서의 속성 중에 NOTFOUND를 언급하였는데 오라클에서는 이외에도 다양한 커서의 속성을 통해 커서의 상태를 알려주는데 이 속성을 이용해서 커서를 제어해야 한다.

속성	의미
%NOTFOUND	커서 영역의 자료가 모두 FETCH됐었다면 TRUE
%FOUND	커서 영역에 FETCH 되지 않은 자료가 있다면 TRUE
%ISOPEN	커서가 OPEN된 상태이면 TRUE
%ROWCOUNT	커서가 얻어 온 레코드의 개수

6.5 CLOSE CURSOR

- CLOSE문장은 CURSOR를 사용할 수 없게 하고 결과 셋의 정의를 해제한다.

- SELECT 문장이 다 처리된 완성 후에는 CURSOR를 닫는다. 필요하다면 CURSOR를 다시 열수도 있다.

형식 **CLOSE cursor_name;**

6.6 부서 테이블의 모든 내용 조회하기(1)

- ex) 커서를 사용하여 부서 테이블의 모든 내용을 출력하시오.
- 1. 다음과 같이 입력하시오.

예

```
CREATE OR REPLACE PROCEDURE CURSOR_SAMPLE01
IS
    VDEPT DEPT%ROWTYPE;
    CURSOR C1
    IS
    SELECT * FROM  DEPT;
BEGIN
    DBMS_OUTPUT.PUT_LINE('부서번호 / 부서명 / 지역명');
    DBMS_OUTPUT.PUT_LINE('----------------------');
    OPEN C1;
```

6.6 부서 테이블의 모든 내용 조회하기(2)

- ex) 커서를 사용하여 부서 테이블의 모든 내용을 출력하시오.
- 1. 다음과 같이 입력하시오.

예

```
    LOOP
        FETCH C1 INTO VDEPT.DEPTNO, VDEPT.DNAME,
VDEPT.LOC;
        EXIT WHEN C1%NOTFOUND;
        DBMS_OUTPUT.PUT_LINE(VDEPT.DEPTNO||
        '  '||VDEPT.DNAME||'  '||VDEPT.LOC);
    END LOOP;
   CLOSE C1;
END;
/
```

6.7 CURSOR와 FOR LOOP

- CURSOR FOR LOOP는 명시적 CURSOR에서 행을 처리한다. LOOP에서 각 반복마다 CURSOR를 열고 행을 인출(FETCH)하고 모든 행이 처리되면 자동으로 CURSOR가 CLOSE되므로 사용하기가 편리하다.

형식	**FOR *record_name* IN *cursor_name* LOOP** ***statement1;*** ***statement2;*** **. . . . . .** **END LOOP**

- OPEN ~ FETCH ~ CLOSE가 없이 FOR ~ LOOP ~ END LOOP문을 사용하여 보다 간단하게 커서를 처리해보자.

6.8 부서 테이블의 모든 내용 조회하기(1)

- ex) 커서를 사용하여 부서 테이블의 모든 내용을 출력하시오.
- 1. 다음과 같이 입력하시오.

예

```
CREATE OR REPLACE PROCEDURE CURSOR_SAMPLE02
IS
    VDEPT DEPT%ROWTYPE;
    CURSOR C1
    IS
    SELECT * FROM DEPT;
BEGIN
    DBMS_OUTPUT.PUT_LINE('부서번호 / 부서명 / 지역명');
    DBMS_OUTPUT.PUT_LINE('--------------------------');
    FOR VDEPT IN C1 LOOP
        EXIT WHEN C1%NOTFOUND;
        DBMS_OUTPUT.PUT_LINE(VDEPT.DEPTNO||
        ' '||VDEPT.DNAME||' '||VDEPT.LOC);
    END LOOP;
END;
/
```