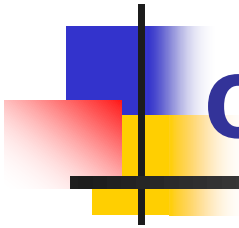
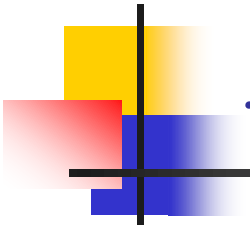


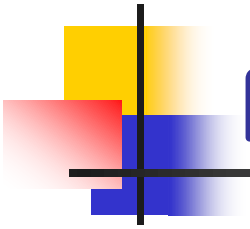
Oracle 9강 – PL/SQL

양 명 숙
[now4ever7@gmail.com]



목차

- PL/SQL 이란
- 함수
- 저장 프로시저
- 커서



PL/SQL 이란?

- Procedural language extension to Structured Query Language
- SQL과 일반 프로그래밍 언어의 특성을 결합한 언어
- 변수, 상수 선언 가능
- 조건문, 반복문 사용 가능

- DBMS의 역할이 커지면서 SQL을 넘어서는 일반 프로그래밍 언어가 처리할 수 있는 기능들이 필요하게 됨
- => 1989년 Oracle 6 버전부터 PL/SQL 이 등장

선언부, 실행부, 예외처리부가 하나의 PL/SQL 블록을 구성하고, 오라클은 이 블록 단위로 처리함

PL/SQL 기본구조

- 선언부 (Declarative part)
  - 실행부에서 사용할 변수나 상수 선언
  - 변수와 상수 선언은 반드시 선언부에서만 선언해야 함
  - Declare 사용
- 실행부 (Executable part)
  - 실제 처리할 로직(logic) 부분 담당
  - 변수에 값 할당, if, while 문장 사용, sql 문장 사용 등
  - PL/SQL의 엔진 역할
  - 실행부에 여러 개의 sql 문장들을 순차적으로 사용하여 블록 단위로 한 번에 처리할 수 있다는 장점
  - Begin으로 시작, End 로 끝난다
- 예외처리부 (Exception-building part)
  - 실행부에서 로직을 처리하던 중 발생 가능한 각종 오류 처리 부분
  - 오라클에서는 sql 문장 실행시 오류가 발생하면 자동으로 ORA-xxxx 오류번호와 함께 메시지가 반환되는데, 사용자가 직접 이러한 메시지를 대체하거나 오류가 발생할 경우 처리할 로직을 기술하는 부분
  - Exception 키워드 사용

PL/SQL 실행

- SQL*PLUS, toad 등에서 실행

- [간단한 PL/SQL Anonymous 블록 예]

```
DECLARE
   counter INTEGER;
BEGIN
   counter := 10;
   counter := counter / 0;
   dbms_output.put_line(counter);
EXCEPTION WHEN OTHERS THEN
   dbms_output.put_line('ERRORS');
END;
```

선언부

실행부

예외처리부

PL/SQL 실행

- 예외처리 구문

  EXCEPTION WHEN 예외1 THEN 예외처리1
  WHEN 예외2 THEN 예외처리2
  ….
  WHEN OTHERS THEN 나머지예외처리

- 미리 정의된 예외 및 사용자 정의 예외처리 가능
- 다른 프로그래밍 언어의 TRY … CATCH 문과 유사

```
-- PL/SQL이란?
DECLARE
   counter INTEGER;
BEGIN
   counter := counter + 1;
   IF counter IS NULL THEN
      dbms_output.put_line('Result : COUNTER IS Null');
   END IF;
END;
```

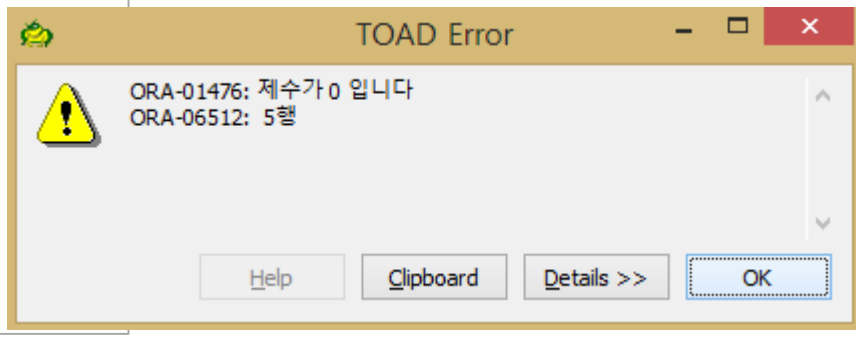
1	Result : COUNTER IS Null

dbms_output.put_line()
- ()안에 명시한 내용을 출력하는 기능의 시스템 패키지

```
DECLARE
   counter INTEGER;
   i  INTEGER;
BEGIN
   FOR i IN 1..10 LOOP
      counter := (2 * i);
      dbms_output.put_line(' 2 * ' || i || ' = ' || counter );
   END LOOP;
END;
```

1	2 * 1 = 2
2	2 * 2 = 4
3	2 * 3 = 6
4	2 * 4 = 8
5	2 * 5 = 10
6	2 * 6 = 12
7	2 * 7 = 14
8	2 * 8 = 16
9	2 * 9 = 18
10	2 * 10 = 20

```
DECLARE
   counter INTEGER;
BEGIN
   counter := 10;
   counter := counter / 0;
   dbms_output.put_line(counter);
END;
```



```
DECLARE
   counter INTEGER;
BEGIN
   counter := 10;
   counter := counter / 0;
   dbms_output.put_line(counter);
EXCEPTION WHEN OTHERS THEN
   dbms_output.put_line('ERRORS');
END;
```

1	ERRORS

```
DECLARE
   counter INTEGER;
BEGIN
   counter := 10;
   counter := counter / 0;
   dbms_output.put_line(counter);
EXCEPTION WHEN ZERO_DIVIDE THEN
   dbms_output.put_line('ERRORS');
END;
```

1	ERRORS

```
DECLARE
   counter INTEGER;
BEGIN
   counter := 10;
   counter := counter / 0;
   dbms_output.put_line(counter);
EXCEPTION WHEN ZERO_DIVIDE THEN
   counter := counter / 1;
   dbms_output.put_line(counter);
END;
```

1	10

예제

```
--[1]
declare  --선언부 : 변수나 상수 선언
    counter  number;
begin --실행부 :  처리할 로직
    counter := 1;
    counter := counter/0;
    if counter is not null then
        dbms_output.put_line('counter 변수의 값 : ' || counter );
    end if;
Exception when ZERO_DIVIDE then
    dbms_output.put_line('0으로 나눌 수 없습니다!');
end;
```

1	0으로 나눌 수 없습니다!

```
--[2]
declare
    counter  number;
    i        number;
begin
    for i  in 1..10 loop
        counter := 2*i;
        dbms_output.put_line(counter);
    end loop;
exception when others then
    dbms_output.put_line('error 발생');
end;
```

1	2
2	4
3	6
4	8
5	10
6	12
7	14
8	16
9	18
10	20

```
--[3]
declare
    counter number;
    i   number;
begin
    i := 1;
    while i<=20 loop
        counter := 2*i;
        dbms_output.put_line('i=' || i || ', counter
                                    =' || counter);
        i := i+1;
    end loop;
 exception when others then
    dbms_output.put_line('error!!');
end;
```

```
i=1, counter=2
i=2, counter=4
i=3, counter=6
i=4, counter=8
i=5, counter=10
i=6, counter=12
i=7, counter=14
i=8, counter=16
i=9, counter=18
i=10, counter=20
i=11, counter=22
i=12, counter=24
i=13, counter=26
i=14, counter=28
i=15, counter=30
i=16, counter=32
i=17, counter=34
i=18, counter=36
i=19, counter=38
i=20, counter=40
```

PL/SQL 미리 정의된 예외

예외 내용	예외 번호	SQLCODE	발생시점
ACCESS_INTO_NULL	ORA-06530	-6530	초기화 되지 않은 오브젝트에 값을 할당하려고 할 경우
CASE_NOT_FOUND	ORA-06592	-6592	CASE 문장에서 ELSE 구문도 없고 WHEN 절에 명시된 조건을 만족하는 것이 하나도 없을 경우
COLLECTION_IS_NULL	ORA-06531	-6531	초기화 되지 않은 중첩 테이블이나 VARRAY같은 콜렉션을 EXISTS 외의 다른 메소드로 접근을 시도할 경우 발생
CURSOR_ALREADY_OPEN	ORA-06511	-6511	이미 오픈된 커서를 다시 오픈하려고 시도하는 경우
DUP_VAL_ON_INDEX	ORA-00001	-1	유일 인덱스가 걸린 컬럼에 중복 데이터를 입력할 경우
INVALID_CURSOR	ORA-01001	-1001	허용되지 않은 커서에 접근할 경우(오픈되지 않은 커서를 닫으려고 시도하는 경우)

INVALID_NUMBER	ORA-01722	-1722	SQL 문장에서 문자형 데이터를 숫자형으로 변환할 때 제대로 된 숫자로 변환이 되지 않을 경우
LOGIN_DENIED	ORA-01017	-1017	잘못된 사용자나 비밀번호로 로그인을 시도할 때
NO_DATA_FOUND	ORA-01403	+100	SELECT INTO 문장의 결과로 선택된 로우가 하나도 없을 경우
NOT_LOGGED_ON	ORA-01012	-1012	오라클에 연결되지 않았을 경우
PROGRAM_ERROR	ORA-06501	-6501	PL/SQL 내부에 문제가 발생했을 경우
SELF_IS_NULL	ORA-30625	-30625	OBJECT 타입이 초기화 되지 않은 상태에서 MEMBER 메소드를 사용할 경우
STORAGE_ERROR	ORA-06500	-6500	메모리가 부족한 경우
SUBSCRIPT_BEYOND_COUNT	ORA-06533	-6533	중첩 테이블이나 VARRAY의 요소값에 접근할 때, 명시한 인덱스 번호가 콜렉션 전체 크기를 넘어설 경우
SUBSCRIPT_OUTSIDE_LIMIT	ORA-06532	-6532	중첩 테이블이나 VARRAY의 요소값에 접근할 때, 잘못된 인덱스 번호를 사용할 경우(예, 인덱스 번호로 -1 사용시)
SYS_INVALID_ROWID	ORA-01410	-1410	문자열을 ROWID로 변환할 때 변환값에 해당하는 ROWID 값이 없을 경우
TIMEOUT_ON_RESOURCE	ORA-00051	-51	오라클이 리소스를 기다리는 동안 타임아웃이 발생했을 때
TOO_MANY_ROWS	ORA-01422	-1422	SELECT INTO 문장에서 하나 이상의 로우가 반환될 때
VALUE_ERROR	ORA-06502	-6502	문자형 데이터를 숫자형으로 변환하는데 타당한 숫자가 아니거나 값을 할당 시 값의 크기가 선언된 변수의 크기를 넘어서는 경우와 같이 값을 변환하거나 할당할 때 오류가 발생할 경우
ZERO_DIVIDE	ORA-01476	-1476	제수가 0일 때 발생

PL/SQL의 구성요소 – 변수와 상수

- 일반적 변수 및 상수 선언과 유사
- 상수 선언 시, 상수명 뒤에 CONSTANT 키워드를 붙여야 함
- %TYPE
  - 기존 테이블에 있는 컬럼의 데이터 타입을 그대로 사용

```
변수명   데이터 타입
상수명  CONSTANT  데이터 타입
변수명   테이블명.컬럼명%TYPE
```

- %ROWTYPE
  - %TYPE과 유사하나, 한 개 이상의 값에 대해 적용
  - 로우타입 변수를 선언해 테이블에 있는 로우 대입 가능
  - 컬렉션이나 object 타입 변수에서만 사용

- 변수 및 상수 선언 예

```
emp_num1 NUMBER(9);
nYear CONSTANT INTEGER := 30;
nSalaries EMP.SAL%TYPE;
```

PL/SQL의 구성요소 – 레코드

- 테이블 형태의 데이터 타입
- 구조체(Structure)와 비슷 (각기 다른 데이터 타입을 갖는다)

- [구문형식]

  – TYPE 레코드이름 IS RECORD (필드1 데이터타입1, 필드2 데이터타입2 …);

  – 레코드이름 테이블명%ROWTYPE;
  – 레코드이름 커서명%ROWTYPE;

```
DECLARE
   -- TYPE으로 선언한 레코드
   TYPE record1 IS RECORD ( dep_id NUMBER NOT NULL := 50, dep_name VARCHAR2(30),
                            loc_id NUMBER );

   -- 위에서 선언한 record1을 받는 변수 선언
   rec1 record1;

   -- 테이블명%ROWTYPE을 이용한 레코드 선언
   rec2 dept%ROWTYPE;

   CURSOR C1 IS
          SELECT deptno, dname, loc
            FROM dept
           WHERE loc = 'DALLAS';

   -- 커서명%ROWTYPE 을 이용한 레코드 선언
   rec3 C1%ROWTYPE;

BEGIN
   -- record1 레코드 변수 rec1의 dep_name 필드에 값 할당.
   rec1.dep_name := '레코드부서1';

   -- rec2 변수에 값 할당
   rec2.deptno := 60;
   rec2.dname := '레코드부서2';
   rec2.loc := 'test';

   -- rec1 레코드 값을 dept 테이블에 INSERT
   INSERT INTO dept  VALUES rec1;
   -- rec2 레코드 값을 dept 테이블에 INSERT
   INSERT INTO dept VALUES rec2;
```

1	20
2	20

```
 -- 커서 오픈
  OPEN C1;

  LOOP
    -- 커서 값을 rec3에 할당한다. 개별 값이 아닌 deptno, dname,
    -- loc 값이 레코드 단위로 할당된다.
    FETCH C1 INTO rec3;
    dbms_output.put_line(rec3.deptno);
    EXIT WHEN C1%NOTFOUND;
  END LOOP;

  COMMIT;
EXCEPTION WHEN OTHERS THEN
            ROLLBACK;
END;
```

PL/SQL 문장

- IF 문 : 조건 처리문

```
IF 조건 THEN
        처리문1;
ELSIF 조건2 THEN
        처리문2;
    ...
ELSE
       처리문2;
END IF;
```

- CASE 문

```
CASE ...
          WHEN ... THEN ...
          ELSE ...
END CASE;
```

PL/SQL 문장

- LOOP 문

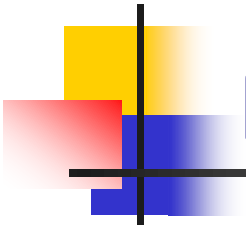
```
LOOP
        처리문장들 ...;
END LOOP;
```

- WHILE-LOOP 문

```
WHILE 조건 LOOP
        처리문장들 ...;
END LOOP;
```

- FOR - LOOP 문

```
FOR 카운터 IN [REVERSE] 최소값..최대값 LOOP
        처리문장들 ...;
END LOOP;
```



PL/SQL 문장

- GOTO 문
  - 특정 위치로 분기하는 문장

- NULL 문
  - 아무것도 처리하지 않는 문장

```
DECLARE
  grade CHAR(1);
BEGIN
  grade := 'B';
  IF grade = 'A' THEN
     dbms_output.put_line('Excellent');
  ELSIF grade = 'B'  THEN
     dbms_output.put_line('Good');
  ELSIF grade = 'C'  THEN
     dbms_output.put_line('Fair');
  ELSIF grade = 'D'  THEN
     dbms_output.put_line('Poor');
  END IF;
END;
```

1	Good

```
DECLARE
  grade CHAR(1);
BEGIN
  grade := 'B';
  CASE grade
     WHEN 'A' THEN
        dbms_output.put_line('Excellent');
     WHEN 'B' THEN
        dbms_output.put_line('Good');
     WHEN 'C' THEN
        dbms_output.put_line('Fair');
     WHEN 'D' THEN
        dbms_output.put_line('Poor');
     ELSE
        dbms_output.put_line('Not Found');
  END CASE;
END;
```

1	Good

```
DECLARE
  test_number INTEGER;
  result_num  INTEGER;
BEGIN
  test_number := 1;

  LOOP
    result_num := 2 * test_number;
    IF result_num > 20 THEN
       EXIT;
    ELSE
       dbms_output.put_line(result_num);
    END IF;
    test_number := test_number + 1;
  END LOOP;
END;
```

1	2
2	4
3	6
4	8
5	10
6	12
7	14
8	16
9	18
10	20

```
DECLARE
  test_number INTEGER;
  result_num  INTEGER;
BEGIN
  test_number := 1;

  LOOP
    result_num := 2 * test_number;

    EXIT WHEN result_num > 20;
    dbms_output.put_line(result_num);
    test_number := test_number + 1;
  END LOOP;
END;
```

1	2
2	4
3	6
4	8
5	10
6	12
7	14
8	16
9	18
10	20

```
DECLARE
  test_number INTEGER;
  result_num  INTEGER;  --null
BEGIN
  test_number := 1;

  WHILE result_num <= 20 LOOP  --null과 비교 => 결과 안나온다

    result_num := 2 * test_number;
    dbms_output.put_line(result_num);
    test_number := test_number + 1;

  END LOOP;
END;

DECLARE
  test_number INTEGER;
  result_num  INTEGER;
BEGIN
  test_number := 1;
  result_num := 0;  -- 초기값  넣기

  WHILE result_num <= 20 LOOP

    result_num := 2 * test_number;
    dbms_output.put_line(result_num);
    test_number := test_number + 1;

  END LOOP;
END;
```

1	2
2	4
3	6
4	8
5	10
6	12
7	14
8	16
9	18
10	20
11	22

```
DECLARE
  test_number INTEGER;
  result_num  INTEGER;
BEGIN
  test_number := 1;
  result_num := 0;

  dbms_output.put_line('<<first>>');
  <<first>>
  FOR test_number IN 1..10 LOOP
    result_num := 2 * test_number;
    dbms_output.put_line(result_num);
  END LOOP;

  dbms_output.put_line('<<second>>');

  result_num := 0;
  <<second>>
  FOR test_number IN REVERSE 1..10 LOOP
    result_num := 2 * test_number;
    dbms_output.put_line(result_num);
  END LOOP;
END;
```

1	<<first>>
2	2
3	4
4	6
5	8
6	10
7	12
8	14
9	16
10	18
11	20
12	<<second>>
13	20
14	18
15	16
16	14
17	12
18	10
19	8
20	6
21	4
22	2

```
DECLARE
  test_number INTEGER;
  result_num  INTEGER;
BEGIN
  test_number := 1;
  result_num := 0;

  GOTO second;
  dbms_output.put_line('<<first>>');
  <<first>>
  FOR test_number IN 1..10 LOOP
    result_num := 2 * test_number;
    dbms_output.put_line(result_num);
  END LOOP;

    <<second>>
  result_num := 0;
  dbms_output.put_line('<<second>>');

  FOR test_number IN REVERSE 1..10 LOOP
    result_num := 2 * test_number;
    dbms_output.put_line(result_num);
  END LOOP;
END;
```

1	<<second>>
2	20
3	18
4	16
5	14
6	12
7	10
8	8
9	6
10	4
11	2

예제

exec test1_proc('C');

```
--[4]
create or replace procedure test1_proc
 (p_grade char)
 is
    v_result varchar2(50);
begin
    if p_grade = 'A' then
        v_result := 'excellent';
    elsif p_grade = 'B' then
        v_result := 'good';
    else
        v_result := 'poor';
    end if;
    dbms_output.put_line(v_result);
 exception when others then
    dbms_output.put_line('error!!');
end;
```

1	poor

exec test1_proc('A');

```
--[5]
 create or replace procedure test1_proc
 (p_grade char)
 is
    v_result varchar2(50);
 begin
    case p_grade
        when 'A' then
            v_result := 'excellent';
        when 'B' then
            v_result := 'good';
        else
            v_result := 'poor';
    end case;
    dbms_output.put_line('p_grade=' ||p_grade ||
',v_result=' || v_result);
 exception when others then
    dbms_output.put_line('error!!');
end;
```

1	p_grade=A,v_result=excellent

예제

```
--[6]
create or replace procedure test1_proc
 is
    i  number;
    result number;
begin
    for i in reverse 1..10 loop
        result := 2*i;
        DBMS_OUTPUT.PUT_LINE(result);
    end loop;
 exception when others then
    dbms_output.put_line('error 발생');
end;
```

1	20
2	18
3	16
4	14
5	12
6	10
7	8
8	6
9	4
10	2

PL/SQL 서브프로그램 – 함수

- PL/SQL 서브프로그램 – 데이터베이스 객체로 저장해서 필요할 때마다 호출하여 사용할 수 있는 PL/SQL 블록
  - 내장 프로시저(stored procedure)
  - 함수(function)
- 함수 – 사용자 정의 함수를 말함
  - 특정 기능을 수행한 뒤, 결과값을 반환하는 서브프로그램
- [구문형식]

```
CREATE OR REPLACE FUNCTION 함수명
            ( 파라미터1 데이터타입,
              파라미터2 데이터타입, ...)
     RETURN 데이터타입
IS [AS]
   BEGIN
       처리내용 ....
       RETURN 반환값;
    END;
```

예제- 성별 구하는 함수

```
create or replace function getGender_func
(p_jumin IN varchar2)
return varchar2
is
   v_gender varchar2(2);
begin
    select case when substr(p_jumin, 7, 1) in('1','3') then '남'
        else '여' end
        into v_gender
    from dual;

    return v_gender;
end;

------
select getGender_func('9501011112222') from dual;
```

예제-나이 구하는 함수

```
create or replace function getAge_func
(p_jumin IN varchar2)
return number
is
   v_age  number(3);
begin
    select extract(year from sysdate) -
        (to_number(substr(p_jumin, 1,2))
         + case when substr(p_jumin, 7, 1) in('1','2') then 1900
            else 2000 end)
        + 1
    into v_age
    from dual;

    return v_age;
end;
----
select getAge_func('9501011112222') from dual;
```

```
select studno, name, grade,
getGender_func(jumin) "성별", getAge_func(jumin) "나이"
from student
WHERE deptno1 = 101;
```

PL/SQL 서브프로그램 – 프로시저

- 특정 기능을 수행하지만 **값을 반환하지는 않는** 서브프로그램
- [구문형식]

```
CREATE OR REPLACE PROCEDURE 프로시저명
            ( 파라미터1 데이터타입,
              파라미터2 데이터타입, ...)
   IS [AS]
       변수선언부 ...
   BEGIN
       처리내용 ....
      EXCEPTION 예외처리 ...
   END;
```

• 프로시저 소스 보기

```
select *  from user_source;
```

예제

```
--실행
EXECUTE pdInsert_proc('B01','마이크',25000, '기가폰');
EXEC  pdInsert_proc('B02','마이크',25000, '기가폰');

select * from pd2;
```

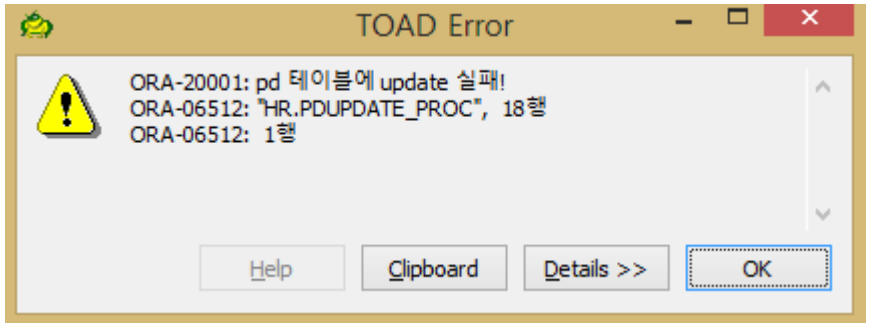
```
--pd2 테이블에 입력하는 프로시저
create or replace procedure pdInsert_proc
(
    p_pdcode     char,
    p_pdname     varchar2,
    p_price      number,
    p_company    varchar2
)
is
begin
    insert into pd2(no, pdcode, pdname, price,company)
    values(pd2_seq.nextval, p_pdcode, p_pdname,
        p_price, p_company);
    commit;

    exception when others then
        dbms_output.put_line('pd2 테이블에 insert 실패!');
        rollback;
end;
```

```
--pd2 테이블 수정 프로시저-특정 no에 해당하는 레코드 수정하기
create or replace procedure pdUpdate_proc(
    p_no            pd2.no%type,
    p_pdcode        pd2.pdcode%type,
    p_pdname        pd2.pdname%type,
    p_price         pd2.price%type
)
is
    v_cnt number;
begin
  select count(*) into v_cnt
  from pd2
  WHERE no=p_no;

  if v_cnt>0 then
    update pd2
    set pdcode=p_pdcode, pdname=p_pdname, price=p_price
    where no=p_no;

    commit;
  end if;

  exception when others then
       dbms_output.put_line('pd2 테이블에 update 실패!');
       rollback;
end;
```

```
exec pdUpdate_proc (5, 'B02', '휴대폰', 780000);

select * from pd2;
```

주의 : 컬럼명과 매개변수 명이 달라야 구분 되어 수정 가능

pd2 테이블 수정 프로시저




```
create or replace procedure pdUpdate_proc
(
    p_no          pd2.no%type,
    p_pdcode      pd2.pdcode%type,
    p_pdname      pd2.pdname%type,
    p_price       pd2.price%type
)
is
begin
    update pd2 A
    set pdcode=p_pdcode, pdname=p_pdname,
        price=p_price
    where exists( select 1 from pd2 B where A.no=B.no and B.no=p_no);

    commit;

  exception when others then
      raise_application_error(-20001, 'pd 테이블에 update 실패!');
      rollback;
end;
```

사용자 정의 예외번호는 -20001 부터 -20999 까지의 범위내에서 만들어야 함.

type 생성

exec professor_info(1001);

```
create or replace procedure professor_info
 (p_profno IN number)
 is
    --- type 을 생성
    type proftype
    is record
    (deptno  professor.deptno%type,
    profno   professor.profno%type,
    name     professor.name%type,
    position  professor.position%type,
    salary     number(10)
    );
    v_prof_record proftype;
 begin
      select deptno, profno , name, position,
            nvl2(bonus, pay+bonus, pay)
           into v_prof_record
      from professor
      where profno = p_profno;

      dbms_output.put_line('학과번호  교수번호 교수명 직위  월급');
      dbms_output.put_line( v_prof_record.deptno || ' ' ||  v_prof_record.profno || ' ' ||v_prof_record.name || ' ' ||
                          v_prof_record.position  || ' ' || v_prof_record.salary);
   exception when others then
       dbms_output.put_line('error!');

 end;
```

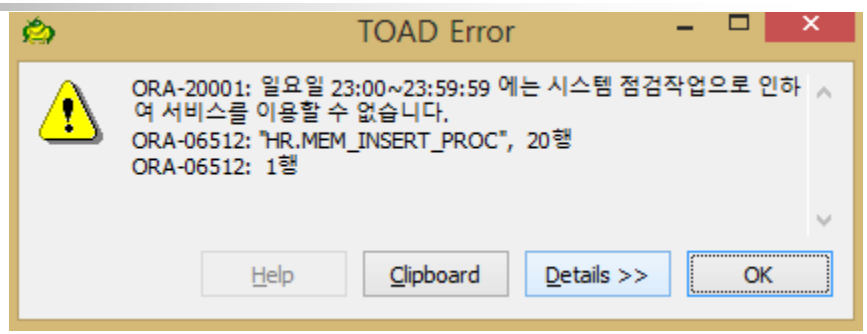
```
create or replace procedure professor_info
 (p_profno IN number)
 is
   v_prof_row  professor%rowtype;
   v_result  varchar2(1000);
 begin
     select * into v_prof_row
     from professor
     where profno = p_profno;

      v_result := v_prof_row.deptno || ' ' ||
                  v_prof_row.profno  || ' ' ||
                  v_prof_row.name || ' ' ||
                  nvl(v_prof_row.pay+ v_prof_row.bonus,
                   v_prof_row.pay) || ' ' ||
                  v_prof_row.position;

     dbms_output.put_line('학과번호 교수번호 교수명 월급 직위');
     dbms_output.put_line( v_result );

 end;
```

%rowtype

```
exec professor_info(1001);
```

exec mem_insert_proc('kim', '1', '김길동', '010-100-2000');

사용자 정의 예외




```
create or replace procedure mem_insert_proc
(p_id IN mem.id%type,
p_pwd IN mem.pwd%type,
p_name IN mem.name%type,
p_hp IN mem.hp%type
)
is
   system_check_insert_fail exception;
begin
      if to_char(sysdate, 'd')= '1' and to_char(sysdate, 'hh24') = '23' then  --1:일요일
         raise system_check_insert_fail;
      end if;

      insert into mem(no, id, pwd, name, hp)
      values(mem_seq.nextval, p_id, p_pwd, p_name, p_hp);
      commit;

      exception
         when system_check_insert_fail then
            raise_application_error(-20001, '일요일 23:00~23:59:59 에는 시스템 점검작업으로 인하여
      서비스를 이용할 수 없습니다.');
end;
```

out 매개변수

```
create or replace procedure infoProf_proc
(p_profno   in professor.profno%type,
o_name  out professor.name%type,
o_pay out professor.pay%type)
is
begin
    select name, pay into o_name, o_pay
    from professor
    where profno=p_profno;
end;

-----------------------------------------

declare
v_name  professor.name%type;
v_pay  professor.pay%type;
begin
    infoProf_proc(1001, v_name, v_pay);

    dbms_output.put_line('이름  :'|| v_name);
    dbms_output.put_line('급여  :'|| v_pay);
end;
```

```
1  이름 :조인형
2  급여 :550
```

PL/SQL 커서

PL/SQL 에서 select 결과물의 레코드 개수가 2 개 이상인 경우 반드시 커서를 이용해야 함

- 커서
  - 쿼리에 의해 반환되는 결과는 메모리 상에 위치하게 되는데, PL/SQL에서는 커서를 사용하여 이 결과집합에 접근할 수 있다.
  - 커서를 사용하면 결과집합의 각 개별 데이터에 접근이 가능하다.
- 명시적 커서
  - 사용자가 직접 쿼리의 결과에 접근해서 이를 사용하기 위해 명시적으로 선언한 커서
  - 명시적 커서를 사용하기 위한 절차
    - [1] 커서 선언 – 쿼리 정의

```
CURSOR 커서명 IS SELECT 문장;
```

    - [2] 커서 열기(open) – 쿼리 실행

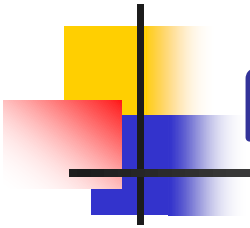
```
OPEN 커서명;
```

    - [3] 패치(fetch) – 쿼리의 결과에 접근, 루프를 돌며 개별 값들에 접근

```
FETCH 커서명 IS 변수 ...;
```

    - [4] 커서 닫기(close) – 메모리상에 존재하는 쿼리의 결과를 소멸시킴

```
CLOSE 커서명;
```



PL/SQL 커서

- 묵시적 커서
  - 오라클 내부에서 각 쿼리 결과에 접근하여 사용하기 위한 내부적 커서
  - 모든 쿼리가 실행될 때마다 오픈됨
  - 오라클 내부에서 접근하고 사용되는 커서이므로 선언, 오픈 등의 작업을 할 필요가 없다
  - 항상 가장 최근에 실행된 sql 문장에 대한 커서를 내부적으로 갖고 있는데, 이를 sql 커서라 하며, 'SQL' 이라는 이름으로 속성에 접근할 수 있다

```
create or replace procedure pdSelect_proc
is
    --[1]커서 선언 - 쿼리 정의
    cursor pd_csr is
        select no, pdcode, pdname from pd2;
    --변수선언
    v_pd    pd2%rowtype;
begin
    --[2] 커서 열기
    open pd_csr;

    --[3] 패치(fetch) - 쿼리의 결과에 접근, 루프를 돌며 개별 값들에 접근
    loop
        fetch pd_csr into v_pd.no, v_pd.pdcode, v_pd.pdname;
        exit when pd_csr %notfound;
        dbms_output.put_line(v_pd.no || ' ' || v_pd.pdcode ||' ' || v_pd.pdname);
    end loop;

--[4] 커서 닫기 - 메모리상에 존재하는 쿼리 결과 집합을 소멸시킴
    close pd_csr;

    exception when others then
        dbms_output.put_line('select error!!');
end;
```

```
exec pdSelect_proc();
```

%ROWTYPE
- %TYPE과 유사하나, 한 개 이상의 값에 대해 적용
- 로우타입 변수를 선언해 테이블에 있는 로우 대입 가능

1	2	B01	마이크
2	6	B03	휴대폰
3	7	B04	컴퓨터

%NOTFOUND – 커서에서만 사용 가능한 속성
- 더 이상 패치(할당)할 로우가 없음을 의미
- 쿼리의 마지막 결과까지 패치한 후에 자동으로 루프를 빠져나가게 됨

```
DECLARE        --묵시적 커서
  count1 NUMBER;
  count2 NUMBER;
BEGIN
  SELECT count(*)
    INTO count1
    FROM emp
   WHERE deptno = 20;

  count2 := SQL%ROWCOUNT;  --묵시적 커서가 패치된 수를 count2 변수에 할당
  dbms_output.put_line('SELECT COUNT IS ' || count1 );
  dbms_output.put_line('ROW COUNT IS ' || count2 );
END;
```

커서의 속성변수

- 1. 커서명%ISOPEN - 커서가 OPEN 상태인가를 체크
  - 커서가 OPEN 되어있다면 true 반환
- 2. 커서명%FOUND - FETCH 된 레코드(행)이 있는지 체크
  - FETCH 된 레코드(행)이 있으면 true 반환
- 3. 커서명%NOTFOUND-FETCH 된 레코드(행)이 없는지 체크
  - FETCH 된 레코드(행)이 없으면 true 를 반환
- 4. 커서명%ROWCOUNT - 현재까지 FETCH 된 레코드(행)의 갯수를 반환

FOR LOOP CURSOR 문

- 커서의 FOR LOOP 문을 사용하면 커서의 OPEN, FETCH, CLOSE 가 자동적으로 발생되어지기 때문에 OPEN, FETCH, CLOSE 문을 기술할 필요가 없다.

- 형식

FOR 변수명(레코드정보가 담기는 변수) IN 커서명 LOOP
    실행문장;
END LOOP;

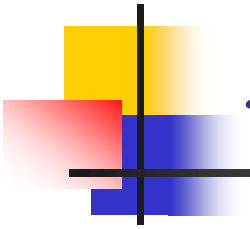
```
create or replace procedure professor_info2
(p_deptno  IN  professor.deptno%type)
is
  cursor cur_prof -- 1단계
  is
  select deptno, profno,
         name,
         nvl2(bonus,pay+bonus, pay) as salary,
         position
  from professor
  where deptno = p_deptno;

  v_count number := 0;
begin
    dbms_output.put_line('학과번호 교수번호 교수명 월급 직위');

    for prof_row  in cur_prof loop  -- 2단계
        dbms_output.put_line(prof_row.deptno || ' ' ||
                             prof_row.profno || ' ' ||
                             prof_row.name || ' ' ||
                             prof_row.salary || ' ' || prof_row.position);
        if(cur_prof%found) then
           v_count := v_count + 1;
        end if;
    end loop;

    dbms_output.put_line(v_count || '건 조회되었습니다');
end;
```

```
exec professor_info2(101);
```

SYS_REFCURSOR

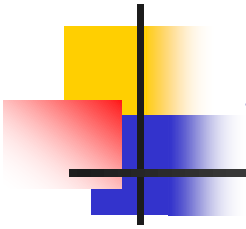
- 저장 프로시저의 select 결과물을 JAVA 에서 읽기 위해서는 SYS_REFCURSOR 타입을 사용해야 함

```
create or replace procedure pdList_proc
(pdCursor out SYS_REFCURSOR)
is
begin
    OPEN pdCursor For
    select * from pd2
    order by no desc;

    exception when others then
       raise_application_error(-20001,'pd select error!!');
end;
```



실습

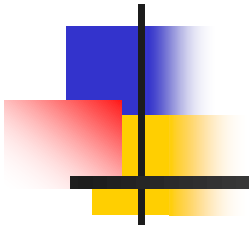
- cart 테이블에 insert 하는 저장 프로시저
- update 프로시저 : cartno로 수정
- delete 프로시저 : cartno로 삭제
- select 프로시저 :userid에 해당하는 내용 조회 => 커서 이용



실습

- 1~20까지의 합 – for문 이용

- if 문 이용
  - 1~3 => 1사분기
  - 4~6 => 2사분기
  - 7~9 => 3사분기
  - 10~12 => 4사분기



누적 합계 구하기

```
--누적 합계 구하기
select p_code as "제품코드",   p_date as "판매일자",     p_qty as "판매량",
       sum(p_qty) over(partition by p_code  order by p_date)    as "누적판매량"
  from panmae;

select p_code, sum(p_qty) from panmae; --error

select A.p_code as "제품코드",   A.p_date as "판매일자",    A.qty_total as "판매량"
sum(A.qty_total) over(partition by A.p_code   order by A.p_date)   as "누적판매량"
 from
 (
  select p_code,   p_date,   sum(p_qty) as qty_total   from panmae
  group by p_code, p_date
 ) A;
 ----------------------
 create or replace view v_panmae
 as
 select A.p_name as "제품명",     A.p_date as "판매일자",    A.qty_total as "판매량",
       sum(A.qty_total) over(partition by A.p_name   order by A.p_date)  as "누적판매량"
 from
 (
  select b.p_name,   p_date,  sum(p_qty) as qty_total
  from panmae p join product b   on p.p_code=b.p_code
  group by b.p_name, p.p_date
 ) A;
```

제품코드	판매일자	판매량	누적판매량
100	20110101	3	3
100	20110103	15	18
100	20110104	11	29
101	20110101	5	5
101	20110103	4	9
101	20110104	10	19
102	20110101	2	2
102	20110102	2	4
102	20110104	6	10
103	20110101	6	6
103	20110102	5	11
103	20110104	2	13
104	20110102	3	3
105	20110102	2	2
106	20110105	3	3
107	20110105	4	4
107	20110106	2	6
108	20110107	2	2

```
--panmae 테이블을 사용하여 판매내역을 출력하되 판매일자, 판매량,판매금액,누적 판매량,
--누적 판매금액을 출력
select sum(p_qty) qty_tot, sum(p_total) total_tot
    from panmae;

select p_date, qty_tot, total_tot,
    sum(qty_tot) over(order by p_date) 누적판매량,
    sum(total_tot) over(order by p_date) 누적판매금액
from(
    select p_date, sum(p_qty) qty_tot, sum(p_total) total_tot
    from panmae
    group by p_date
);
```

P_DATE	QTY_TOT	TOTAL_TOT	누적판매량	누적판매금액
20110101	16	14300	16	14300
20110102	12	11900	28	26200
20110103	19	15600	47	41800
20110104	29	25600	76	67400
20110105	7	6700	83	74100
20110106	2	2000	85	76100
20110107	2	1800	87	77900