

PL/SQL

가. PL/SQL 이란?

PL/SQL: Oracle's Procedural Language extension to SQL

오라클에 내장되어 있는 절차형 언어

프로그램을 논리적인 블록으로 나누는 구조화된 블록 언어

변수 선언문, 조건문, 반복문(loop, while, for) 등을 지원

나. PL/SQL Block Structure

선언부(선택), 실행부(필수), 예외 처리부(선택)로 구성됨

begin 과 end 키워드는 반드시 기술

PL/SQL BLOCK의 기본 구성을 살펴보자

DECLARE : 모든 변수나 상수를 선언하는 선언부

BEGIN : 제어문, 반복문, 함수 정의 등을 기술하는 실행부

Exception : 실행 도중 에러 발생 시 해결하는 문장을 기술하는 예외처리부

(선언부와 예외처리부는 경우에 따라 생략 가능)

1) Declareative Section(선언부, 선택)

변수, 상수, Cursor등을 선언

2) Executable Section(실행부, 필수)

sql문장, 반복문, 조건문 등을 실행

begin으로 시작하고 end로 끝남

3) Exception Handling Section(예외처리부, 선택)

예외에 대한 처리

다. Block type(pl/sql 블록의 유형)

[Anonymous]

[Procedure]

[Function]



1) Anonymous (익명 블록)

이름이 없는 블록

2) Procedure(프로시저)

특정 작업을 수행할 수 있고, 이름이 있는 PL/SQL 블록

매개 변수를 입력받을 수 있음

매개 변수를 입력받을 수 있음

DB에 저장되어 반복적으로 사용할 수 있음 (저장 프로시저)

배치 작업(일괄처리) 또는 구현이 복잡한 트랜잭션을 수행하는 용도로 사용함

ex) 어떤 건에 대해서 자료를 입력하는데 기존에 자료가 있으면 insert를 하고, 기존에 자료가 있으면 update를 한다

-> count함수를 통해 레코드의 개수를 세어서 레코드 개수가 0이면 insert하고 레코드 개수가 1이면 update 한다

-> 이렇게 복잡한 sql을 프로시저 내에서 한꺼번에 처리가능 하다

3) Function(함수)

값을 계산하고 결과값을 반환하기 위해서 사용

저장 프로시저와의 차이점: 입력매개변수만 사용 할 수 있고 리턴 타입을 반드시 지정해야 함

프로시저- 입력매개변수와 출력매개변수가 있음

라. 저장프로시저

1) Stored Procedure(SP, 저장 프로시저)

가) 특정 작업을 수행할 수 있고, 이름이 있는 PL/SQL블록

나) 매개 변수를 입력받을 수 있음

다)DB에 저장되어 반복적으로 사용할 수 있음

라) 배치 작업 또는 구현이 복잡한 트랜잭션을 수행하는 용도로 사용함

2) 형식

```
CREATE OR REPLACE 저장프로시저이름  
(매개변수)  
IS  
    변수 선언  
BEGIN  
    문장  
END;
```

3) 저장 프로시저 실습 예제

가) 급여 인상 저장 프로시저 실습

=====

--create or replace procedure 프로시저이름(매개변수)

--입력매개변수: 변수명 in 자료형

--출력매개변수: 변수명 out 자료형

=====

CREATE OR REPLACE PROCEDURE UPDATE_SAL

(V_EMPNO IN NUMBER) --입력매개변수

is--선언부

BEGIN --실행부

update employees

set salary=salary*1.1

where employee_id=v_empno;

commit;

END;

/

=====

--저장 프로시저의 실행 방법

--execute 저장프로시저이름 (입력값)

execute update_sal(100);

select *from employees ;

=====

--자바에서 프로시저 실행

```
Connection con = null;
CallableStatement cstmt = null;

try {
    for (DetailCodeTO Code : saveCodeList) {
        con = dataSourceTransactionManager.getConnection();
        StringBuffer query = new StringBuffer();
        query.append("{call P_SAVE_CODE(?,?,?,?)}");
        cstmt = con.prepareCall(query.toString());
        cstmt.setString(1, Code.getDetailCodeNumber());
        cstmt.setString(2, Code.getCodeNumber());
        cstmt.setString(3, Code.getDetailCodeName());
        cstmt.setString(4, Code.getDetailCodeNameusing());
        cstmt.setString(5, Code.getStatus().toUpperCase());

        cstmt.executeUpdate();

    }
}
```

```
1 CREATE OR REPLACE PROCEDURE UPDATE_SAL
2 (V_EMPNO IN NUMBER ) --입력매개변수
3 is--선언부
4 BEGIN --실행부
5     update employees
6     set salary=salary*1.1
7     where employee_id=v_empno;
8     commit;
9 END;
10
11 / --pl/sql블록을 실행
12
```

select *from employees ;

크립트 출력 x | 질의 결과 x

SQL | 50개의 행이 인출됨(0.004초)

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY	COMMISSION_PCT	MANAGER_ID	DEPARTMENT_ID
1	Steven	King	SKING	515.123.4567	87/06/17	AD_PRES	24000	(null)	(null)	
2	Neena	Kochhar	NKOCHHAR	515.123.4568	89/09/21	AD_VP	17000	(null)	100	
3	Lex	De Haan	LDEHAAN	515.123.4569	93/01/13	AD_VP	17000	(null)	100	
4	Alexander	Hunold	AHUNOLD	590.423.4567	90/01/03	IT_PROG	9000	(null)	102	
5	Bruce	Ernst	BERNST	590.423.4568	91/05/21	IT_PROG	6000	(null)	103	

로그

```
execute update_sal(100);
```

```
select * from employees;
```

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY	COMMISSION_PCT	MANAGER_ID	DEPARTMENT_ID
100	Steven	King	SKING	515.123.4567	87/06/17	AD_PRES	26400	(null)	(null)	
101	Neena	Kochhar	NKOCHHAR	515.123.4568	89/09/21	AD_VP	17000	(null)	100	
102	Lex	De Haan	LDEHAAN	515.123.4569	93/01/13	AD_VP	17000	(null)	100	
103	Alexander	Hunold	AHUNOLD	590.423.4567	90/01/03	IT_PROG	9000	(null)	102	
104	Bruce	Ernst	BERNST	590.423.4568	91/05/21	IT_PROG	6000	(null)	103	

나) 한 줄 메모장 저장 프로시저 실습

=====

----한 줄 메모장 테이블

```
create table memo(
```

```
idx number primary key, --기본키
```

```
writer varchar2(50) not null,
```

```
memo varchar2(500) not null,
```

```
post_date date default sysdate—기본값 현재시간
```

```
)
```

-----시퀀스 생성

```
create sequence memo_seq
```

```
start with 1 --부터 시작
```

```
increment by 1 --씩 증가
```

```
nomaxvalue; --무제한 증가
```

insert into memo

values(memo_seq.nextval,'kim','memo')

--저장 프로시저 작성

```
create or replace procedure
memo_insert( v_writer in varchar, v_memo in varchar, v_ip in varchar )
is
begin
    insert into memo (idx,writer,memo,ip)
    values ( memo_seq.nextval, v_writer, v_memo, v_ip);
end;
/
```

--저장 프로시저 호출

```
select * from memo;
--out 출력매개변수(호출한 곳으로 리턴되는 값)
--sys_refcursor : 레코드를 한개씩 조회할 수 있는 자료형(커서)
-- jdbc의 ResultSet과 비슷한 역할
create or replace procedure memo_list(v_row out sys_refcursor)
is
begin
    open v_row for
        select idx,writer,memo,post_date,ip
        from memo
        order by idx desc;
end;
/
```

다) 커서를 이용해서 hr계정의 employees테이블에서 정보 뽑기 실습

```

1 CREATE OR REPLACE PROCEDURE EMP_LIST
2 (v_emp_code out sys_refcursor)
3 is
4 BEGIN
5     open v_emp_code for
6     select first_name from employees;
7 END;

```

CREATE OR REPLACE PROCEDURE EMP_LIST

(v_emp_code out sys_refcursor)

is

BEGIN

open v_emp_code for

select first_name from employees;

END;

실행:

```

var a refcursor
exec EMP_LIST(:a)
print a

```

var a refcursor

exec EMP_LIST(:a)

print a

=====

마. 함수(Function)

1) 값을 계산하고 결과값을 반환하기 위해서 사용

2) 저장 프로시저와의 차이점

입력 매개변수만 사용할 수 있고 리턴 타입을 반드시 지정해야 함

(return이 반드시 필요한데 한개 밖에 안됨)

3) 형식

```
CREATE OR REPLACE FUNCTION 함수이름  
[(argument...)]  
RETURN datatype  
-- Datatype 은 반환되는 값의 datatype 입니다.  
IS  
    변수 선언  
BEGIN  
    문장  
    return  
END;
```

create or replace functuon 함수이름

[(argument...)]

return datatype

--Datatype은 반환되는 값의 datatype 입니다

IS

변수선언

Begin

문장

return

End;

4) 함수 예제

```
create or replace function fn_update_sal(v_empno number)
return number --리턴 자료형
is
    v_sal number; -- 지역변수
begin
    update emp set sal=sal*1.1 where empno=v_empno;
    -- select sal*1.1 into v_sal from emp --10% 인상금액을 v_sal에 저장
    select sal into v_sal from emp
    where empno=v_empno;
    return v_sal; --인상된 금액을 리턴
end;
/
```

```
--7369번 사원의 급여
select sal from emp where empno=7369;
--10% 인상된 급여, dual (가상테이블)
select fn_update_sal(7369) from dual;
select empno,ename,sal,sal*1.1, fn_update_sal(empno) from emp;
--dual : 가상테이블(테스트목적)
--desc 테이블 : 테이블에 대한 설명
desc dual;
select sysdate from dual;
```

```
20▶ create or replace function fn_update_sal
21    (v_empno number) -- 입력매개변수만 허용됨
22    return number -- 리턴 자료형
23    is
24        v_sal number; -- 지역변수
25    begin
26        update emp set sal=sal*1.1 where empno=v_empno;
27        select sal into v_sal from emp
28        where empno=v_empno;
29
30        return v_sal; -- 리턴값
31    end;
32    /
```

create or replace function fn_update_sal(v_empno number) --입력 매개변수만 허용됨

return number

is

v_sal number;

begin

update employees set salary=salary*1.1 where employee_id=v_empno;

select salary into v_sal from employees

where employee_id=v_empno;

return v_sal; --- 리턴값

end;

/

```
select * from emp where empno=7369;
--ORA-14551: cannot perform a DML operation
--inside a query
-- 함수를 select 문장에서 실행할 때는 DML 금지
select fn_update_sal(7369) from dual;
```

실행해보기:

```
select fn_update_sal(100) from dual;
--ORA-14551: cannot perform a DML operation inside a query
--inside a query
-- 함수를 select 문장에서 실행할 때는 dml 금지
```

```
select *from employees;
```

```
select *from employees where employee_id=100;
```

```
select fn_update_sal(100) from dual;
```

```
--DML(Data Manipulation Language, 데이터 조작 언어-insert,delete,update)
-- 질의 안에 DML 작업을 수행할 수 없습니다
-- "cannot perform a DML operation inside a query "
```

```
--변수 선언
--함수 호출, 리턴값을 salary변수에 저장
--salary변수에 저장된 값을 출력
-- A := B => B의 값을 A에 대입
-- :변수 => 바인딩변수(리턴값을 받는 변수)
var salary number;
execute :salary := fn_update_sal(7369);
print salary;
```

```
11 |
12 | var sal number;
13 | execute :sal :=fn_update_sal(100);
14 | print sal;
15 |
.. |
```

```
var sal number;
```

```
execute :sal :=fn_update_sal(100);
```

```
print sal;
```

함수 안에서는 update delete 같은 문장이 사용가능 하지만 최대한 사용x 하는것이 좋다고 함

```
--> 저장 프로시저 사용
```

바. pl/sql 제어문

--> pl/sql에도 제어문이 있음 ex) if문 for문 while문 등등

1) %type 테이터형

가) 테이블의 컬럼 데이터 타입을 모를 경우 사용

나) 테이블의 데이터 타입이 변경될 경우 다시 수정할 필요가 없음

다) 예제

%type을 사용하게 되면 우리가 자료형을 구체적을 적지 않아도 그 테이블의 자료형으로 처리가 가능

```
-- 테이블.컬럼%type
create or replace procedure emp_info(p_empno in emp.empno%type )
is --변수 선언
  v_empno emp.empno%type; -- 테이블.컬럼%type
  v_ename emp.ename%type;
  v_sal emp.sal%type;
begin
  select empno,ename,sal into v_empno,v_ename,v_sal
  from emp
  where empno=p_empno;
  -- dbms_output 패키지의 put_line 함수 호출
  dbms_output.put_line('사번:'||v_empno); --pl/sql 출력문
  dbms_output.put_line('이름:'||v_ename);
  dbms_output.put_line('급여:'||v_sal);
end;
/
set serveroutput on
execute emp_info(7369);
```

자바에서 패키지란? 여러가지 클래스들을 묶은 것

오라클에서 패키지란? 여러가지 함수나 프로시저들을 묶은 것

```

create or replace procedure employees_info
(p_employee_id in employees.employee_id%type)
is
    v_employee_id employees.employee_id%type;
    v_first_name employees.first_name%type;
    v_salary employees.salary%type;
begin
    select employee_id, first_name, salary into v_employee_id, v_first_name, v_salary
    from employees
    where employee_id = p_employee_id;
    --패키지.함수
    dbms_output.put_Line('사번:' || v_employee_id);
    dbms_output.put_Line('이름:' || v_first_name);
    dbms_output.put_Line('급여:' || v_salary);

end;
/
set serveroutput on
execute employees_info(100);

```

create or replace procedure employees_info

(p_employee_id in employees.employee_id%type)

is

 v_employee_id employees.employee_id%type;

 v_first_name employees.first_name%type;

 v_salary employees.salary%type;

begin

 select employee_id, first_name, salary into v_employee_id, v_first_name, v_salary

 from employees

 where employee_id = p_employee_id;

 --패키지.함수

 dbms_output.put_Line('사번:' || v_employee_id);

 dbms_output.put_Line('이름:' || v_first_name);

 dbms_output.put_Line('급여:' || v_salary);

end;

/

set serveroutput on

execute employees_info(100);

2) if 문

가) 형식



나) if문 예제

```

create or replace procedure dept_search(p_empno in number)
is
    v_deptno number;
begin
    select deptno into v_deptno from emp
    where empno=p_empno;
    dbms_output.put_line('부서코드:'||v_deptno);
    if v_deptno = 10 then -- := 대입 = 비교
        dbms_output.put_line('경리팀 직원입니다');
    elsif v_deptno = 20 then
        dbms_output.put_line('연구팀 직원입니다');
    elsif v_deptno = 30 then
        dbms_output.put_line('총무팀 직원입니다');
    else
        dbms_output.put_line('기타부서 직원입니다');
    end if;
end;

```

pl/sql 에서 A=B 같다, A:=B 대입

3) for loop 문

가) index: 자동 선언되는 binary_integer 형 변수 1씩 증가

나) reverse 옵션이 사용되 경우 1씩 감소

다) in 다음에는 cursor나 select 문이 올수 있음

```

FOR index IN [REVERSE] 시작값 .. END값 LOOP
    statement 1
    statement 2
    ....
END LOOP;

```


ex)

```
for cnt in 1 .. i loop
  dbms_output.put('이름:||' || ename_tab(cnt)||',');
  dbms_output.put_line('급여:||' || sal_tab(cnt));
end loop;
end;
/
select * from emp;
```

라) 예제

-- binary_integer: pl/sql에서 사용하는 정수형 변수(java의 int)

-- type 사용자정의 자료형 is table of 테이블

-- index by 배열의 인덱스 지정

declare -- 선언부

-- 사용자 정의 자료형

```

--무병 블록
declare --선언부
    --type 자료형 이름 is ...

    type ename_table
        is table of employees.employee_id%type index by binary_integer;
    type sal_table
        is table of employees.salary%type index by binary_integer;

    --변수명 자료형
    ename_tab ename_table;
    sal_tab sal_table;
    i binary_integer := 0;
begin
    -- for 레코드 변수 in 집합
    for emp_row in (select employee_id, salary from employees) loop
        i := i+1;
        ename_tab(i) := emp_row.employee_id;
        sal_tab(i) := emp_row.salary;
    end loop;
    --for 카운트 변수 in 시작... 끝
    for cnt in 1 .. i loop
        dbms_output.put('이름: ' || ename_tab(cnt) || ', ');
        dbms_output.put_line('급여: ' || sal_tab(cnt));
    end loop;
end;

```

--무병 블록

declare --선언부

--type 자료형 이름 is ...

type ename_table

is table of employees.employee_id%type index by binary_integer;

type sal_table

is table of employees.salary%type index by binary_integer;

--변수명 자료형

ename_tab ename_table;

sal_tab sal_table;

```

i binary_integer := 0;

begin
-- for 레코드 변수 in 집합

for emp_row in (select employee_id, salary from employees) loop

i := i+1;

ename_tab(i) := emp_row.employee_id;

sal_tab(i) := emp_row.salary;

end loop;

--for 카운트 변수 in 시작... 끝

for cnt in 1 .. i loop

dbms_output.put('이름:|| ename_tab(cnt)||','');

dbms_output.put_Line('급여:|| sal_tab(cnt));

end loop;

end;

```

4) loop문

가)exit: 무조건 loop문 종료

나) exit when: loop문 빠져 나가는 조건을 제어



```
-- dbms_output.put_line 함수가 작동되도록 처리
set serveroutput on
select * from emp where empno >=9010;
delete from emp where empno >=9010;
declare --변수 선언부
    v_cnt number := 9010;
begin
    loop
        v_cnt := v_cnt+1;
        insert into emp (empno,ename,hiredate) values
            (v_cnt, 'test'||v_cnt, sysdate);
        exit when v_cnt >= 9100; --루프 종료 조건
    end loop;
    dbms_output.put_line(v_cnt-9010||'개의 레코드가 입력되었습니다.');
```

5) while loop문

가) for문과 비슷하며 조건이 true 일 경우만 반복되는 loop

나) 예제

```
-- dbms_output.put_line 함수가 작동되도록 처리
set serveroutput on
declare --변수 선언부
    cnt number := 9050;
begin
    while cnt < 9060 loop
        insert into emp(empno, ename , hiredate)
        values (emp_seq.nextval, 'test', sysdate);
        cnt := cnt + 1 ;
    end loop ; --루프 종료 조건
```