14강 PL/SQL

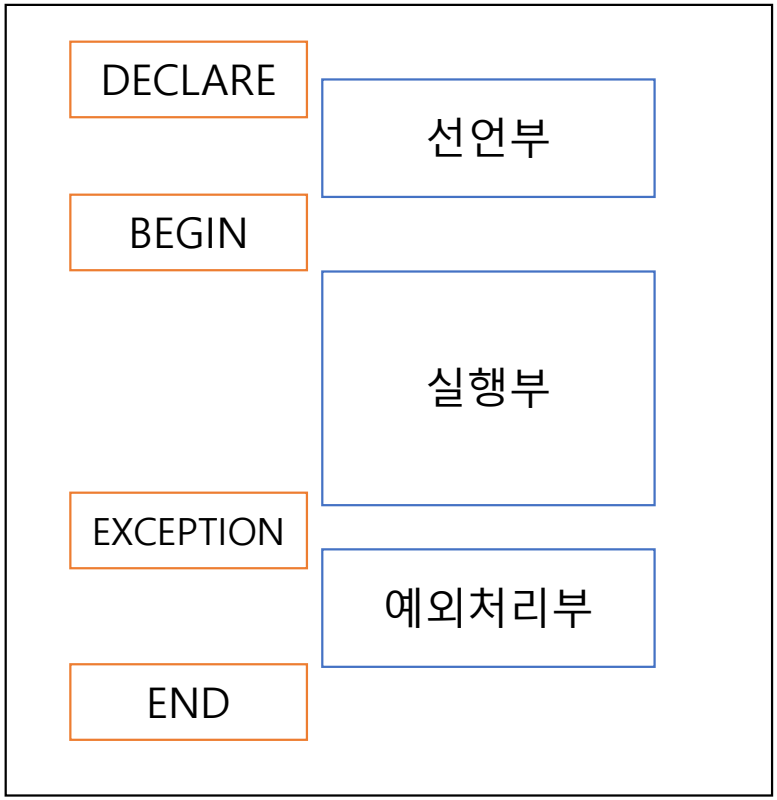
PL/SQL 의 개념

- PL/SQL(Oracle's Procedural Language extension to SQL)은 오라클에서 지원하는 프로그래밍 언어의 특성을 수용해서 기존 SQL에서는 사용할 수 없는 절차적 프로그래밍 기능으로 SQL의 단점을 보완하기 위해 만든 언어이다.

  - 변수와 상수를 선언하여 SQL과 절차적 언어에서 사용
  - IF문을 사용하여 조건에 따라 문장을 분기
  - LOOP문을 사용하여 일련의 문장을 반복적으로 실행
  - 커서를 사용하여 여러 행을 검색

PL/SQL 의 개념

- 기본 구조



PL/SQL 의 개념

- 각 구조의 설명

SECTION	설 명	필수여부
DECLARE (선언부)	PL/SQL에서 사용하는 모든 변수나 상수를 선언하는 부분 DECLARE로 시작	옵션
BEGIN (실행부)	절차적 형식으로 SQL문을 실행할 수 있도록 절차적 언어의 요소인 제어문, 반복문, 함수 정의 등 로직을 기술할 수 있는 부분 BEGIN으로 시작	필수
EXCEPTION (예외처리부)	PL/SQL문이 실행되는 중에 에러가 발생할 수 있는데 이를 예외 사항이다. 이런 예외 사항이 발생했을 때 이를 해결하기 위한 문장을 기술하는 부분 EXCEPTION으로 시작	옵션

PL/SQL 의 개념

- 작성 요령
  - PL/SQL 블록 내에서는 한 문장이 종료할 때마다 세미콜론(;)을 쓴다.
  - END 뒤에는 ;을 사용하여 하나의 블록이 끝났음을 명시
  - PL/SQL 블록의 작성은 편집기를 통해 파일로 작성할 수도 있고, 프롬프트에서 바로 작성 가능
  - SQL*PLUS환경에서는 DECLARE나 BEGIN이라는 키워드로 PL/SQL 의 시작을 알린다.
  - 블록 내부에서 단일 행 주석은 -- 다중 행 주석은 /* */ 이다
  - 쿼리문 끝에 /가 붙으면 종결된 것으로 간주

- 반드시 출력하기 위해서는 환경변수 SERVEROUTPUT 을 ON으로 바꾸어 주어야 한다.(워크시트당 한번만 실행)

```
SET serveroutput ON;
```

PL/SQL 의 개념

- 기본 작성 예제

```
BEGIN
    dbms_output.put_line('Hello World!');
END;
```

변수 선언

- PL/SQL 블록 내부에 변수를 사용하려면 선언부(DELCARE)에서 선언해야 하며 변수명 다음에 데이터 타입을 기술해야 한다.

  - 형식 : identifier datatype := expression
    변수명 타입 표현식

  - 변수명에 CONSTANT를 붙이면 상수로서 작동 => 값을 변경할 수 없다.
  - 타입 뒤에 NOT NULL을 붙이면 반드시 초기값을 지정하도록 제약한다.

  - 보통 한중에 하나의 변수를 정의한다.

변수 선언

- 기본적으로 사용되는 데이터 타입은 스칼라와 레퍼런스 타입으로 나뉜다.

- 스칼라 : SQL에서 사용하던 데이터 타입과 일치
  - 숫자, 문자, 날짜 BOOLEAN 4가지

- 레퍼런스 : 기존 칼럼에 사용된 타입을 선언하기 위해서 사용하는 타입
  - 테이블명.컬럼명%type => 해당 컬럼 타입 가져오기
  - 테이블명%rowtype => 해당 레코드 전체 타입 가져오기

  - 개발자는 테이블에 정의된 컬럼의 타입이나 크기를 알수 없으므로 유용하다

변수 선언

- 대입문 (:=) => 변수 := 값

- 변수에 값 할당하는 예제

```
DECLARE
    v_eno number(4);
    v_ename employee.ename%type;
BEGIN
    v_eno := 7788;
    v_ename :='SCOTT';
    dbms_output.put_line('사원번호   사원이름');
    dbms_output.put_line('----------------------');
    dbms_output.put_line(v_eno||'     '||v_ename);
END;
```

변수 선언

- PL/SQL에서도 테이블의 조회를 위해서 SELECT문을 사용한다.
- 다만 일반 SQL 문과 차이점은 INTO라는 절이 추가적으로 필요한다.

- INTO절의 역할은 조회된 데이터를 저장하기 위한 변수를 연결하기 위함이다.

변수 선언

- 예제 : 사원이름이 SCOTT인 사원의 사번과 이름을 출력하세요.

```
DECLARE
    v_eno employee.eno%type;
    v_ename employee.ename%type;
BEGIN
    dbms_output.put_line('사원번호    사원이름');
    dbms_output.put_line('----------------------');

    SELECT eno,ename INTO v_eno, v_ename
    FROM employee
    WHERE ename='SCOTT';

    dbms_output.put_line(v_eno||'      '||v_ename);
END;
```

제어문

- IF문(다른 언어의 조건문과 같다)
  - 조건의 값이 참과 거짓을 판단해서 실행문을 실행할지 판단하는 문장이다.
  - ELSIF 절은 여러 번
  - ELSE는 한번만 사용가능

```
IF 조건 THEN
   실행문;

ELSIF 조건 THEN
   실행문;
ELSIF 조건 THEN
   실행문;
ELSIF 조건 THEN
   실행문;
ELSE
    실행문;
END IF;
```

제어문

- 예제: 커미션을 받는 직원은 급여에 12를 곱한후 커미션과 합산하여 연봉을 구하고 커미션이 없는 직원은 급여에 12을 곱한 것으로 연봉을 구한다.

```
DECLARE
    annsal number :=0;
    v_salary employee.salary%type;
    v_commission employee.commission%type;
BEGIN
    SELECT salary,commission INTO v_salary,v_commission
    FROM employee
    WHERE eno<8000 AND ename='SCOTT';
    IF v_commission is NULL THEN
        annsal := v_salary*12;
    ELSE
        annsal := v_salary*12+v_commission;
    END IF;
    dbms_output.put_line('연봉 : '||annsal);
END;
```

제어문

- 예제: SCOTT사원의 사원번호와 소속된 부서명을 출력한다.

```
DECLARE
    v_eno employee.eno%type;
    v_ename employee.ename%type;
    v_dno employee.dno%type;
    v_dname varchar2(20):=null;
BEGIN
    SELECT eno,ename, dno
    INTO v_eno,v_ename,v_dno
    FROM employee
    WHERE ename='SCOTT';

    IF v_dno=10 then
        v_dname:='ACCOUNTING';
    ELSIF v_dno=20 then
        v_dname:='RESEARCH';
    ELSIF v_dno=30 then
        v_dname:='SALES';
    ELSIF v_dno=40 then
        v_dname:='OPERATIONS';
    END IF;

    dbms_output.put_line('사원번호    사원이름    부서명');
    dbms_output.put_line('-----------------------------------------------------------');
    dbms_output.put_line(v_eno||'      '||v_ename||'      '||v_dname);
END;
```

제어문

- LOOP문이란 : 어떤 반복적으로 수행되는 문장이 있을 경우 LOOP 문을 사용합니다
- LOOP문의 종류 : BASIC LOOP문, FOR LOOP문, WHILE LOOP문

제어문

• BASIC LOOP문은 가장 간단한 형태의 루프문으로 LOOP~END LOOP사이를 반복한다.

• 형태

```
LOOP
   실행문1
   실행문2
    ....
    EXIT [WHERE  조건]
END LOOP
```

제어문

- FOR LOOP문은 반복 횟수가 정해진 반복문을 처리한다.
- 수행할 반복횟수를 정하기 위해 LOOP앞에 제어문을 갖는다.
- 형태

```
FOR 인덱스 카운터
IN [R]하단 바운스값 .. 상단 바운스값
LOOP
   실행문1
   실행문2
    ....
END LOOP
```

구문	설명
Index_counter	반복을 위해 선언된 변수(기본은 1 증가값)
REVERSE	증가를 반대로 감소시키는 키워드
lower_bound	최소값
Upper_bound	최대값

제어문

- WHILE LOOP문은 제어 조건이 참인 동안만 문장을 반복한다.
- 조건은 반복이 시작될 때 체크하므로 LOOP내의 문장이 한번도 수행 되지 않을 수 있다

- 형태

```
WHERE 조건
LOOP
   실행문1
   실행문2
    ....
END LOOP
```

제어문

- 예제 : 각 루프문으로 구구단 2단을 구해본다.

```
DECLARE
    dan number:=2;
    i number:=1;
BEGIN
    loop
        dbms_output.put_line(
            dan||'*'||i||'='||(dan*i));
        i:=i+1;
        if i>9 then
            exit;
        end if;
    end loop;
END;
```

```
DECLARE
    dan number:=2;
    i number:=1;
BEGIN
    for i in 1..9 loop
        dbms_output.put_line(
            dan||'*'||i||'='||(dan*i));
    end loop;
END;
```

```
DECLARE
    dan number:=2;
    i number:=1;
BEGIN
    while i<=9 loop
        dbms_output.put_line(
            dan||'*'||i||'='||(dan*i));
        i:=i+1;
    end loop;
END;
```

커서

- SELECT문의 수행 결과가 여러 개의 레코드로 이루어진 경우 각각 레코드에 대한 처리를 하기 위해서 커서를 사용해야 한다.

- 커서는 CURSOR, OPEN, FETCH, CLOSE 4단계 명령에 의해서 사용된다.

커서

- 커서의 형식

```
DECLARE
        CURSOR 커서이름  IS select 쿼리
BEGIN
        OPEN 커서이름

        FECTCH 커서이름 INTO 저장할 변수

        CLOSE 커서이름
END;
```

- 커서의 상태

속성	의미
%NOTFOUND	커서 영역의 자료가 모두 FETCH 되었다면 TRUE
%FOUND	커서 영역에 FETCH 되지 않은 자료가 있다면 TRUE
%ISOPEN	커서가 OPEN 된 상태이면 TRUE
%ROWCOUNT	커서가 얻어온 레코드의 개수

커서

- 커서 사용
  - 1단계 : 커서의 선언
    CURSOR 커서이름
    IS SELECT문(INTO미포함)
  - 2단계 : 커서 오픈
    OPEN 커서이름
  - 3단계 : FETCH
    FETCH 커서이름 INTO (변수1,변수2,...)
    ⇒주로 반복문을 통해서 여러 개의 레코드를 FETCH한다.
    ⇒커서이름%NOTFOUND 조건으로 반복을 종료한다.
  - 4단계 : 커서 종료
    CLOSE 커서이름

커서

- 커서로 department 테이블의 모든 데이터를 조회해본다.

```
DECLARE
    v_dept department%rowtype;
    cursor c1
    is
    SELECT * FROM department;
BEGIN
    dbms_output.put_line(
            '부서번호 부서명     지역명');
    dbms_output.put_line(
            '------------------------------------');
    open c1;
    loop
        fetch c1 into v_dept.dno, v_dept.dname, v_dept.loc;
        exit when c1%notfound;
        dbms_output.put_line(v_dept.dno||'    '
                ||v_dept.dname||'     '||v_dept.loc);
    end loop;
    close c1;
END;
```

커서

- for루프문은 명시적으로 커서를 처리한다.
  - 즉 open, fetch, close없이 FOR 루프문 내부에서 처리를 한다.

```
DECLARE
    v_dept department%rowtype;
    cursor c1
    is
    SELECT * FROM department;
BEGIN
    dbms_output.put_line(
            '부서번호 부서명     지역명');
    dbms_output.put_line(
            '------------------------------------');
    for v_dept in c1 loop
        exit when c1%notfound;
        dbms_output.put_line(v_dept.dno||'    '
                ||v_dept.dname||'     '||v_dept.loc);
    end loop;
END;
```