ch16. PL/SQL 기초

16-1. PL/SQL 구조

블록이란?

PL/SQL은 데이터베이스 관련 특정 작업을 수행하는 명령어와 실행에 필요한 여러 요소를 정의하는 명령어 등으로 구성된다.

- 블록(block): 이러한 명령어를 모아 둔 PL/SQL 프로그램의 기본 단위

구성 키워드	필수 / 선택	설명
DECLARE(선언부)	선택	실행에 사용될 변수, 상수, 커서 등을 선언
BEGIN(실행부)	필수	조건문, 반복문, SELECT, DML, 함수 등을 정의
EXCEPTION(예외 처리부)	선택	PL/SQL 실행 도중 발생하는 오류(예외 상황)를 해결하는 문장 기술

⇒ 작성을 끝낸 PL/SQL 은 END 키워드로 종료를 명시한다.

- PL/SQL 블록의 기본 형식

```
DECLARE
  [ 실행에 칠요한 여러 요소 선언];
BEGIN
  [작업을 위해 실제 실행하는 명령어];
EXCEPTION
  [PL/SQL 수행 도중 발생하는 오류 처리];
END;
```

• 선언부와 에외 처리부는 생략 가능하지만 실행부는 반드시 존재해야한다.

• 중첩 블록(nested block): 필요에 따라 PL/SQL 블록 안에 다른 블록을 포함하는 것

Hello, PL/SQL 출력하기

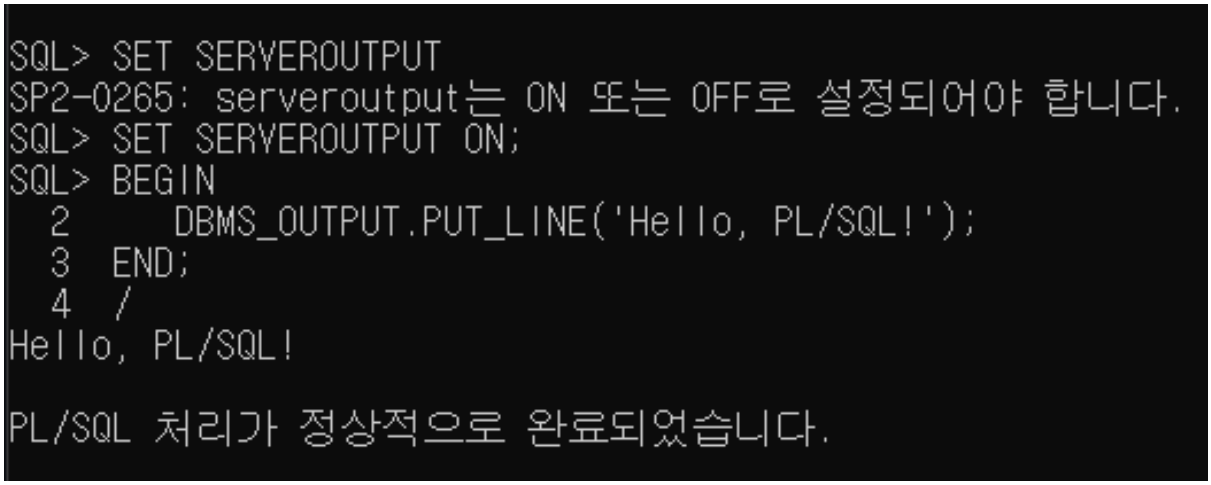
- PL/SQL 실행 결과를 화면에 출력하기 위해 SERVEROUTPUT 환경 변수 값을 ON 으로 변경해주어야 한다.

- PUT_LINE은 화면 출력을 위해 오라클에서 기본으로 제공하며 DBMS_OUTPUT 패키지에 속해있다.

- Hello PL/SQL 출력하기

```
$ SET SERVEROUTPUT ON; -- 실행 결과를 화면에 출력
```

```
$ BEGIN
   DBMS_OUTPUT.PUT_LINE('Hello, PL/SQL!');
END;
/
```



- PL/SQL문을 작성하고 실행하기 위한 사항
  1. PL/SQL 블록을 구성하는 DECLARE, BEGIN, EXCEPTION 키워드에는 세미콜론(;)을 사용하지 않는다.
  2. PL/SQL 블록의 각 부분에서 실행해야하는 문장 끝에는 세미클론(;)을 사용한다.
  3. PL/SQL문 내부에서 한 줄 주석(--)과 여러 줄 주석(/**/)을 사용할 수 있다. 그리고 이들 주석은 SQL문에서도 사용할 수 있다.
  4. PL/SQL문 작성을 마치고 실행하기 위해 마지막에 슬래시(/)를 사용한다.

PL/SQL 주석

- PL/SQL 주석 : PL/SQL 코드에 포함되어 있지만 실행되지 않는 문장을 뜻한다.
- 실행 결과에 아무 영향을 미치지 못하는 문장을 만들기 위해 사용한다.

- 일반적으로 특정 기호를 사용하여 코드 설명 또는 이력 등을 남겨 놓거나, 일시적으로 실행되지 않기를 원하는 코드를 삭제하지 않고 남겨두는 용도로 주석 영역을 지정한다.

종류	사용 기호	설명
한 줄 주석	- - [주석 처리 내용]	현재 줄만 주석 처리됩니다.
여러 줄 주석	/* [주석 처리 내용] */	여러 줄에 걸쳐 주석 처리 됩니다.

- 한줄 주석 사용하기

```
SQL> DECLARE
  2     V_EMPNO NUMBER(4) := 7788;
  3     V_ENAME VARCHAR2(10);
  4  BEGIN
  5     V_ENAME := 'SCOTT';
  6     -- DBMS_OUTPUT.PUT_LINE('V_EMPNO : ' || V_EMPNO);
  7     DBMS_OUTPUT.PUT_LINE('V_ENAME : ' || V_ENAME);
  8  END;
  9  /
V_ENAME : SCOTT

PL/SQL 처리가 정상적으로 완료되었습니다.
```

- 여러 줄 주석 사용하기

```
SQL> DECLARE
  2     V_EMPNO NUMBER(4) := 7788;
  3     V_ENAME VARCHAR2(10);
  4  BEGIN
  5     V_ENAME := 'SCOTT';
  6  /*
  7     DBMS_OUTPUT.PUT_LINE('V_EMPNO : ' || V_EMPNO);
  8     DBMS_OUTPUT.PUT_LINE('V_ENAME : ' || V_ENAME);
  9  */
 10  END;
 11  /

PL/SQL 처리가 정상적으로 완료되었습니다.
```

16-2. 변수와 상수

변수 선언과 값 대입하기

- 변수 *variable* : 데이터를 일시적으로 저장하는 요소
- 변수의 이름과 저장할 자료형을 지정하여 선언부 (DECLARE) 에서 작성한다.
- 선언부에서 작성한 변수는 실행부 (BEGIN) 에서 활용한다.

기본 변수 선언과 사용

```
변수 이름 자료형 := 값 또는 값이 도출되는 여러 표현식;
```

변수 이름 ⇒ 데이터를 저장할 변수 이름을 지정. 이 변수 이름을 통해 저장한 데이터를 사용.

자료형 ⇒ 선언한 변수에 저장할 데이터의 자료형을 지정.

:= ⇒ 선언한 변수에 값을 할당하기 위해 사용. 이 기호는 오른쪽 값을 왼쪽 변수에 대입
하겠다는 뜻. 값을 할당하지 않고 변수 선언만 한다면 :=과 여러 표현식은 생략 가능.

여러 표현식 ⇒ 변수에 저장할 첫 데이터 값이나 저장할 수 있는 값이 결과로 반환되는 표현식을
지정. 이 값은 변수에 지정한 자료형과 맞아야 한다.

- 변수 선언 및 변수 값 출력하기

```
DECLARE
   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;
/
```

상수 정의하기

저장한 값이 필요에 다라 변하는 변수와 달리, 상수는 한 번 저장한 값이 프로그램이 종료될 때까지 유지되는 저장 요소이다.

상수를 선언할 때 다음과 같이 기존 변수 선언에 CONSTANT 키워드를 지정한다.

```
변수 이름 CONSTANT 자료형 := 값 또는 값을 도출하는 여러 표현식
```

⇒ CONSTANT 는 선언한 변수를 상수로 정의하는 것으로 한번 저장한 값은 변하지 않는다.

- 상수에 값을 대입한 후 출력하기

```
DECLARE
   V_TAX CONSTANT NUMBER(1) := 3;
BEGIN
   DBMS_OUTPUT.PUT_LINE('V_TEX : ' || V_TAX);
END;
/
```

변수 기본값 지정하기

DEFAULT 키워드는 변수에 저장할 기본값을 지정한다.

```
변수 이름 DEFAULT 자료형 := 값 또는 값을 도출하는 여러 표현식
```

- 변수에 기본값 설정한 후 출력하기

```
DECLARE
   V_DEPTNO NUMBER(2) DEFAULT 10;
BEGIN
   DBMS_OUTPUT.PUT_LINE('V_DEPTNO : ' || V_DEPTNO);
END;
/
```

변수에 NULL 값 저장 막기

- 특정 변수에 NULL이 저장되지 않게 하려면 NOT NULL 키워드를 사용한다.
- PL/SQL 에서 선언한 변수는 특정 값을 할당하지 않으면 NULL 값이 기본으로 할당된다.
- 이러한 이유로 NOT NULL 키워드를 사용한 변수는 반드시 선언과 동시에 특정 값을 지정해야 함.

```
변수 이름 NOT NULL 자료형 := 또는 DEFAULT 값 또는 값을 도출하는 여러 표현식
```

⇒ := 를 사용하여 선언과 동시에 값을 할당하고 DEFAULT 키워드를 사용하여 변수의 기본값으로
설정할 수도 있다.

- 변수에 NOT NULL을 설정하고 값을 대입한 후 출력하기

```
DECLARE
   V_DEPTNO NUMBER(2) NOT NULL := 10;
BEGIN
   DBMS_OUTPUT.PUT_LINE('V_DEPTNO : ' || V_DEPTNO);
END;
/
```

- 변수에 NOT NULL 및 기본값을 설정한 후 출력하기

```
DECLARE
   V_DEPTNO NUMBER(2) NOT NULL DEFAULT 10;
BEGIN
   DBMS_OUTPUT.PUT_LINE('V_DEPTNO : ' || V_DEPTNO);
END;
/
```

변수 이름 정하기

⇒ 변수를 포함한 SQL/PL문에서 지정하는 객체 이름은 식별자(identifier)라고 한다.

- 식별자에 이름 붙이는 규칙
  1. 같은 블록 안에서 식별자는 고유해야 하며 중복될 수 없다.
  2. 대, 소문자를 구별하지 않는다.
  3. 테이블 이름 붙이는 규칙과 같은 규칙을 따른다.
     a. 이름은 문자로 시작해야 함.
     b. 이름은 30byte 이하여야 함.
     c. 이름은 영문자, 숫자, 특수문자($, #, _) 사용 가능.
     d. SQL키워드는 테이블 이름으로 사용할 수 없음.

변수의 자료형

- 변수에 저장할 데이터가 어떤 종류인지를 특정 짓기 위해 사용하는 자료형은 크게 스칼라 *scaler*, 복합 *composite*, 참조 *reference*, LOB *Large OBject*로 구분된다.

스칼라형

- 스칼라형은 숫자, 문자열, 날짜 등과 같이 오라클에서 기본으로 정의해 놓은 자료형으로 내부 구성 요소가 없는 단일 값을 의미한다.
- 스칼라형은 숫자, 문자열, 날짜, 논리데이터로 나뉜다.

분류	자료형	설명
숫자	NUMBER	소수점을 포함할 수 있는 최대 38자리 숫자 데이터
문자열	CHAR	최대 32,767바이트 고정 길이 문자열 데이터
문자열	VARCHAR2	최대 32,767바이트 가변 길이 문자열 데이터
날짜	DATE	기원전 4712년 1월 1일부터 서기 9999년 12월 31일까지 날짜 데이터
논리 데이터	BOOLEAN	PL/SQL 에서만 사용할 수 있는 논리 자료형으로 true, false,NULL을 포함

참조형

- 참조형은 오라클 데이터베이스에 존재하는 특정 테이블 열의 자료형이나 하나의 행 구조를 참조하는 자료형이다.
- 열을 참조할 때 → %TYPE 사용
- 행을 참조할 때 → %ROWTYPE 사용
- %TYPE으로 선언한 변수는 지정한 테이블 열과 완전히 같은 자료형이 된다.

- %TYPE 사용법

```
변수 이름 테이블이름.열이름%TYPE;
```

- %ROWTYPE 사용법

```
변수 이름 테이블이름.열이름%ROWTYPE;
```

- 참조형(열)의 변수에 값을 대입한 후 출력하기

```
DECLARE
   V_DEPTNO DEPT.DEPTNO%TYPE := 50;
BEGIN
   DBMS_OUTPUT.PUT_LINE('V_DEPTNO : ' || V_DEPTNO);
END;
/
```

- 참조형(행)의 변수에 값을 대입한 후 출력하기

```
DECLARE
   V_DEPT_ROW DEPT%ROWTYPE;
BEGIN
   SELECT DEPTNO, DNAME, LOC INTO V_DEPT_ROW
     FROM DEPT
    WHERE DEPTNO = 40;
   DBMS_OUTPUT.PUT_LINE('DEPTNO : ' || V_DEPT_ROW.DEPTNO);
   DBMS_OUTPUT.PUT_LINE('DNAME : ' || V_DEPT_ROW.DNAME);
```

```
    DBMS_OUTPUT.PUT_LINE('LOC : ' || V_DEPT_ROW.LOC);
END;
/
```

⇒ V_DEPTNO_ROW 변수는 내부에 DEPTNO, DNAME, LOC 필드를 가지게 된다.

복합형, LOB형

- 스칼라형과 참조형 외에도 PL/SQL 에서는 복합형과 LOB형을 사용할 수 있다.
- 복합형 : 여러 종류 및 개수의 데이터를 저장하기 위해 사용자가 직접 정의하는 자료형

분류	자료형	설명
컬렉션	TABLE	한 가지 자료형의 데이터를 여러 개 저장 (테이블의 열과 유사)
레코드	RECORD	여러 종류 자료형의 데이터를 저장 (테이블의 행과 유사)

⇒ Large Object를 의미하는 LOB형은 대용량의 텍스트, 이미지, 동영상, 사운드 데이터 등 대용량 데이터를 저장하기 위한 자료형으로 대표적으로 BLOB, CLOB 등이 있다.

16-3. 조건 제어문

- 특정 조건식을 통해 상황에 따라 실행할 내영을 달리하는 방식의 명령어를 조건문이라고 한다.

IF 조건문

IF-THEN	특정 조건을 만족하는 경우 작업 수행

IF-THEN-ELSE	특정 조건을 만족하는 경우와 반대 경우에 각각 지정한 작업 수행
IF-THEN-ELSIF	여러 조건에 따라 각각 지정한 작업 수행

IF-THEN

- 주어진 조건식의 결과 값이 true인 경우에는 작업을 수행
- false 또는 NULL일 경우에는 작업을 수행하지 않고 다음 내용 실행

```
IF 조건식 THEN
  수행할 명령어;
END IF;
```

IF-THEN-ELSE

- 지정한 조건식의 결과 값이 true일 경우에 실행할 명령어와 조건식의 결과 값이 true가 아닐 대 실행할 명령어를 각각 지정 가능

```
IF 조건식 THEN
  수행할 명령어;
ELSE
  수행할 명령어;
END IF;
```

IF-THEN-ELSIF

- 여러 종류의 조건을 지정하여 각 조건을 만족하는 경우마다 다른 작업의 수행을 지정하는 것이 가능하다.

```
IF 조건식 THEN
  수행할 명령어;
ELSIF 조건식
  수행할 명령어;
ELSIF 조건식
  수행할 명령어;
...
ELSE
  수행할 명령어;
END IF;
```

CASE 조건문

- 단순 CASE문 : 비교 기준이 되는 조건의 값이 여러 가지일 때 해당 값만 명시하여 작업 수행
- 검색 CASE문 : 특정한 비교 기준 없이 여러 조건식을 나열하여 조건식에 맞는 작업 수행

단순 CASE

비교 기준 (여러 가지 결과 값이 나올 수 있는) 이 되는 변수 또는 식을 명시하고, 결과 값에 따라 수행할 작업을 지정.

```
CASE 비교 기준
  WHEN 값1 THEN
    수행할 명령어;
  WHEN THEN
    수행할 명령어;
...
ELSE
    수행할 명령어;
END CASE;
```

검색 CASE

비교 기준을 명시하지 않고 각각의 WHEN절에서 조건식을 명시한 후 해당 조건을 만족할 때 수행할 작업을 정해준다.

```
CASE
  WHEN 조건식1 THEN
    수행할 명령어;
  WHEN 조건식2 THEN
    수행할 명령어;
...
ELSE
    수행할 명령어;
END CASE;
```

- SQL 문의 CASE문은 조건에 따라 특정 결과 값을 반환
- PL/SQL문의 CASE 조건문은 조건에 따라 수행할 작업을 지정할 수 있음.

- SQL문의 CASE는 END로 종료, PL/SQL문의 CASE 조건문은 END CASE로 종료

16-4. 반복 제어문

- 반복문은 특정 작업을 반복하여 수행하고자 할 때 사용한다.

기본 LOOP	기본 반복문
WHILE LOOP	특정 조건식의 결과를 통해 반복 수행
FOR LOOP	반복 횟수를 정하여 반복 수행
Cusor FOR LOOP	커서를 활용한 반복 수행

- 반복 수행을 중단 시키거나 특정 반복 주기를 건너뛰는 명령어

EXIT	수행 중인 반복 종료
EXIT-WHEN	반복 종료를 위한 조건식을 지정하고 만족하면 반복 종료
CONTINUE	수행 중인 반복의 현재 주기를 건너뜀
CONTINUE-WHEN	특정 조건식을 지정하고 조건식을 만족하면 현재 반복 주기를 건너뜀

기본 LOOP

반복을 위한 별다른 조건 없이 반복할 작업 내용을 지정

```
LOOP
  반복 수행 작업;
END LOOP;
```

- 기본 LOOP 사용하기

```
DECLARE
  V_NUM NUMBER := 0;
BEGIN
  LOOP
    DBMS_OUTPUT.PUT_LINE('현재 V_NUM : ' || V_NUM);
    V_NUM := V_NUM + 1;
    EXIT WHEN V_NUM > 4;
  END LOOP;
END;
/
```

WHILE LOOP

반복 수행 여부를 결정하는 조건식을 먼저 지정한 후 조건식의 결과 값이 true일 때 조건을 반복하고 false 가 되면 반복을 끝낸다.

```
WHILE 조건식 LOOP
  반복 수행 작업;
END LOOP;
```

- WHILE LOOP 사용하기

```
DECLARE
  V_NUM NUMBER := 0;
BEGIN
  WHILE V_NUM < 4 LOOP
    DBMS_OUTPUT.PUT_LINE('현재 V_NUM : ' || V_NUM);
    V_NUM := V_NUM + 1;
  END LOOP;
END;
/
```

FOR LOOP

반복하는 횟수를 지정할 수 있는 반복문

```
FOR i IN 시작값...종료값 LOOP
  반복 수행 작업;
END LOOP;
```

- FOR LOOP 사용하기

```
BEGIN
  FOR i IN 0..4 LOOP
    DBMS_OUTPUT.PUT_LINE('현재 i의 값 : ' || i);
  END LOOP;
END;
/
```

- 시작 값에서 종료 값을 역순으로 반복하고 싶다면 REVERSE 키워드를 사용

```
FOR i IN REVERSE 시작값...종료값 LOOP
  반복 수행 작업;
END LOOP;
```

```
BEGIN
  FOR i IN REVERSE 0..4 LOOP
    DBMS_OUTPUT.PUT_LINE('현재 i의 값 : ' || i);
  END LOOP;
END;
/
```

CONITNUE문, CONTINUE-WHEN문

- 반복 수행 중 CONTINUE 가 실행되면 현재 반복 주기에 수행해야 할 남은 작업을 건너뛰고 다음 반복 주기로 바로 넘어가는 효과가 있다.
- CONTINUE는 즉시 다음 반복 주기로 넘어간다.
- CONTINUE-WHEN문은 특정 조건식을 만족할 때 다음 반복 주기로 넘어간다.

- FOR LOOP 안에 CONTINUE문 사용하기

```
BEGIN
  FOR i IN 0..4 LOOP
    CONTINUE WHEN MOD(i , 2) = 1;
    DBMS_OUTPUT.PUT_LINE('현재 i의 값 : ' || i);
  END LOOP;
END;
/
```

Q

1.

```
BEGIN
   FOR i IN 1..10 LOOP
      CONTINUE WHEN MOD(i, 2) = 0;
      DBMS_OUTPUT.PUT_LINE('현재 i의 값 : ' || i);
      END LOOP;
END;
/
```

2.

```
DECLARE
   V_DEPTNO DEPT.DEPTNO%TYPE := 10;
BEGIN
   CASE V_DEPTNO
      WHEN 10 THEN DBMS_OUTPUT.PUT_LINE('DNAME : ACCOUNTING');
      WHEN 20 THEN DBMS_OUTPUT.PUT_LINE('DNAME : RESEARCH');
      WHEN 30 THEN DBMS_OUTPUT.PUT_LINE('DNAME : SALES');
      WHEN 40 THEN DBMS_OUTPUT.PUT_LINE('DNAME : OPERATIONS');
      ELSE         DBMS_OUTPUT.PUT_LINE('DNAME : N/A');
   END CASE;
END;
/
```