

# PL/SQL 배우기

넷째마당에서는 SQL문만으로 해결하기 어려운 작업을 수행하기 위해 오라클에서 제공하는 PL/SQL 프로그래밍 언어에 대해 간략히 알아보겠습니다. 일반적으로 PL/SQL은 데이터 구조뿐만 아니라 데이터를 사용하는 업무 지식 및 용도를 알고 있어야 활용이 가능한데요. 이러한 이유로 데이터베이스를 활용하는 업무에 합류한 지 얼마 되지 않았다면 SQL문보다 PL/SQL문의 사용 빈도가 다소 낮을 것입니다. 이 책에서 소개하고 있는 PL/SQL의 기본 개념과 사용 방법을 시작으로 자신의 업무에 적합한 PL/SQL 사용법을 익혀 나갑시다.

- 16 PL/SQL 기초
- 17 레코드와 컬렉션
- 18 커서와 예외 처리
- 19 저장 서브프로그램

## PL/SQL 기초

PL/SQL은 SQL만으로는 구현이 어렵거나 구현 불가능한 작업을 수행하기 위해 오라클에서 제공하는 프로그래밍 언어입니다. 변수·조건 처리·반복 처리 등 다른 프로그래밍 언어에서도 제공하는 다양한 기능을 사용할 수 있는데요. 이 장에서는 PL/SQL 작성 방법과 실행 등 기본 사용 방법을 알아보겠습니다. 무료로 제공되는 토드 제품 버전에 따라 PL/SQL 사용에 제한이 있으므로 SQL\*PLUS를 통해 예제를 확인해 주세요.

### 16-1 PL/SQL 구조

### 16-2 변수와 상수

### 16-3 조건 제어문

### 16-4 반복 제어문

#### 이 장에서 꼭 익혀야 할 것

- 블록 구조
- 변수 선언 방법
- IF 조건문 사용 방법
- 반복문 사용 방법

# 16-1 PL/SQL 구조

## 블록이란?

PL/SQL은 데이터베이스 관련 특정 작업을 수행하는 명령어와 실행에 필요한 여러 요소를 정의하는 명령어 등으로 구성되며, 이러한 명령어를 모아 둔 PL/SQL 프로그램의 기본 단위를 블록(block)이라고 합니다.

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/SQL은 END 키워드로 종료를 명시합니다.

위 구성을 기반으로 작성한 PL/SQL 블록의 기본 형식은 다음과 같습니다.

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

기본 형식

선언부와 예외 처리부는 생략 가능하지만 실행부는 반드시 존재해야 합니다. 필요에 따라 PL/SQL 블록 안에 다른 블록을 포함할 수도 있습니다. 이를 중첩 블록(nested block)이라고 합니다.

## Hello, PL/SQL 출력하기

먼저 가장 간단한 형태의 PL/SQL문을 작성하여 실행해 보겠습니다. 살펴볼 예제는 SQL\*PLUS에서 Hello, PL/SQL! 문장을 화면에 출력해 보는 예제입니다.

PL/SQL 실행 결과를 화면에 출력하기 위해 다음과 같이 SERVEROUTPUT 환경 변수 값을 ON으로 변경해 주어야 합니다. PUT\_LINE은 화면 출력을 위해 오라클에서 기본으로 제공하며 DBMS\_OUTPUT 패키지에 속해 있습니다. 작성한 PL/SQL문에는 선언부와 예외 처리부가 생략되어 있고 마지막에 슬래시(/)가 작성되어 있음을 눈여겨보세요.

실습 16-1 Hello PL/SQL 출력하기

• 완성 파일 16-1.sql

```
01  SET SERVEROUTPUT ON; -- 실행 결과를 화면에 출력
02  BEGIN
03    DBMS_OUTPUT.PUT_LINE('Hello, PL/SQL!');
04  END;
05 /
```

:: 결과 화면

```
SQL> SET SERVEROUTPUT ON;
SQL> BEGIN
2   DBMS_OUTPUT.PUT_LINE('Hello, PL/SQL!');
3 END;
4 /
Hello, PL/SQL!
PL/SQL 처리가 정상적으로 완료되었습니다.
SQL>
```

이 장의 예제는 SQL\*PLUS에서 실행합니다.

위 PL/SQL문의 실행 결과를 참고하여 PL/SQL문을 작성하고 실행하기 위해 다음 사항을 기억하세요.

1. PL/SQL 블록을 구성하는 DECLARE, BEGIN, EXCEPTION 키워드에는 세미콜론(:)을 사용하지 않습니다.
2. PL/SQL 블록의 각 부분에서 실행해야 하는 문장 끝에는 세미콜론(:)을 사용합니다.  
ex) DBMS\_OUTPUT.PUT\_LINE('Hello, PL/SQL!');
3. PL/SQL문 내부에서 한 줄 주석(--과 여러 줄 주석(\* ~ \*))을 사용할 수 있습니다. 그리고 이들 주석은 SQL문에서도 사용할 수 있습니다.
4. PL/SQL문 작성을 마치고 실행하기 위해 마지막에 슬래시(/)를 사용합니다.

☞ 토드 같은 응용 프로그램에서는 슬래시(/)를 사용하지 않고 PL/SQL문을 실행할 수 있습니다.

그리고 PL/SQL문을 작성하고 실행하기 전에 SET SERVEROUTPUT ON 명령어를 실행하여 PL/SQL문 결과를 화면에 출력하는 것을 잊지 마세요. SQL\*PLUS 접속 설정이 변경되거나 접속이 끊어진 후 다시 SQL\*PLUS를 실행하여 접속했다면 SET SERVEROUTPUT ON 을 반드시 다시 실행해 주어야 PL/SQL 결과를 확인할 수 있습니다.

☞ 이 장 이후에 진행할 모든 PL/SQL문은 SET SERVEROUTPUT ON 명령어가 실행 중임을 가정하고 진행합니다.

## PL/SQL 주석

PL/SQL 주석은 PL/SQL 코드에 포함되어 있지만 실행되지 않는 문장을 뜻합니다. 그러니까 실행 결과에 아무 영향을 미치지 못하는 문장을 만들기 위해 사용하는 거죠. 일반적으로 특정 기호를 사용하여 코드 설명 또는 이력 등을 남겨 놓거나 일시적으로 실행되지 않기를 원하는 코드를 삭제하지 않고 남겨 두는 용도로 주석 영역을 지정합니다. PL/SQL에서 사용하는 주석은 다른 여러 프로그래밍 언어와 마찬가지로 한 줄 주석과 여러 줄 주석으로 나뉩니다.

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

실습 16-2의 실행 코드에서 V\_EMPNO를 출력하는 문장을 다음과 같이 한 줄로 주석 처리해 보죠. 해당 문장은 주석으로 처리되어 실행되지 못하므로 결과에 V\_EMPNO가 출력되지 않습니다.

### 실습 16-2 한 줄 주석 사용하기

• 완성 파일 16-2.sql

```
01  DECLARE
02  V_EMPNO NUMBER(4) := 7788;
03  V_ENAME VARCHAR2(10);
04  BEGIN
05  V_ENAME := 'SCOTT';
06  -- DBMS_OUTPUT.PUT_LINE('V_EMPNO : ' || V_EMPNO);
07  DBMS_OUTPUT.PUT_LINE('V_ENAME : ' || V_ENAME);
08  END;
09 /
```

:: 결과 화면  
V\_ENAME : SCOTT



한 줄 주석은 --를 사용합니다.

이번에는 여러 줄 주석을 사용하여 V\_EMPNO, V\_ENAME을 출력하는 두 줄의 문장을 모두 주석 처리해 보죠. /\* 기호로 시작하여 \*/ 기호로 끝나는 범위 안에 모든 문장은 실행되지 않으므로 결과로 아무것도 출력되지 않습니다.

### 실습 16-3 여러 줄 주석 사용하기

- 완성 파일 16-3.sql

```
01  DECLARE
02    V_EMPNO NUMBER(4) := 7788;
03    V_ENAME VARCHAR2(10);
04
05    BEGIN
06      V_ENAME := 'SCOTT';
07      DBMS_OUTPUT.PUT_LINE('V_EMPNO : ' || V_EMPNO);
08      DBMS_OUTPUT.PUT_LINE('V_ENAME : ' || V_ENAME);
09    */
10  END;
11  /
```



여러 줄 주석은 /\*으로 시작해서 \*/로 끝납니다.

주석은 이처럼 실행하지 않으려는 문장을 지정하는 데 사용하며 PL/SQL문 외에 앞에서 살펴본 SQL문에서도 그대로 사용할 수 있습니다.

1  
분  
복습

다음 빈칸을 채우며 복습해 보세요.

PI /SQL에서 여러 명령어를 모아 두 프로그램의 기본 단위를 <sup>1</sup> 블이라고 합니다.

기본 PL/SQL은 실행에 사용할 변수·상수·커서 등을 선언하는  
언어로 조건문·반복문·SELECT·DML·함수 등 실제 수행할 기능부를 정의하는  
언어로, 그리고 PL/SQL으로 제작한 프로그램의 실행 도중 발생하는 오류를 처리  
하는 언어로 구성됩니다.

## 1. 런ぬ 2. DECLARE 3. BEGIN 4. EXCEPTION

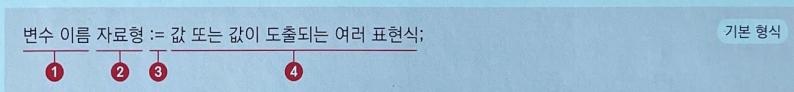
## 16-2 변수와 상수

### 변수 선언과 값 대입하기

변수(variable)는 데이터를 일시적으로 저장하는 요소로 이름과 저장할 자료형을 지정하여 선언부(DECLARE)에서 작성합니다. 선언부에서 작성한 변수는 실행부(BEGIN)에서 활용합니다.

#### 기본 변수 선언과 사용

변수를 선언하는 기본 형식은 다음과 같습니다.



번호	설명
①	데이터를 저장할 변수 이름을 지정합니다. 이 변수 이름을 통해 저장한 데이터를 사용하게 됩니다.
②	선언한 변수에 저장할 데이터의 자료형을 지정합니다.
③	선언한 변수에 값을 할당하기 위해 :=를 사용합니다. 이 기호는 오른쪽 값을 왼쪽 변수에 대입하겠다는 뜻입니다. 값을 할당하지 않고 변수 선언만 한다면 ③, ④는 생략할 수 있습니다.
④	변수에 저장할 첫 데이터 값이나 저장할 수 있는 값이 결과로 반환되는 표현식을 지정합니다. 이 같은 변수에 지정한 자료형과 맞아야 합니다.

그러면 변수를 선언하고 값을 할당한 후 변수에 저장된 값을 출력하는 다음 PL/SQL문을 실행해 보죠. PL/SQL문에서 자료형을 지정하는 방법은 테이블 생성 방식과 비슷합니다. 작은 따옴표(')로 묶인 문자열 데이터와 변수 값을 함께 출력할 때 SELECT문에서 열 데이터 간 연결을 위해 사용한 || 연산자도 눈여겨보세요.

#### 실습 16-4 변수 선언 및 변수 값 출력하기

• 완성 파일 16-4.sql

```
01  DECLARE
02      V_EMPNO NUMBER(4) := 7788;
03      V_ENAME VARCHAR2(10);
04  BEGIN
05      V_ENAME := 'SCOTT';
```

```

06  DBMS_OUTPUT.PUT_LINE('V_EMPNO : ' || V_EMPNO);
07  DBMS_OUTPUT.PUT_LINE('VENAME : ' || VENAME);
08 END;
09 /

```

:: 결과 화면

```

V_EMPNO : 7788
VENAME : SCOTT

```



|| 연산자는 데이터 사이를 연결  
하여 출력할 때 사용합니다.

01행 DECLARE 선언부를 시작합니다.

02행 V\_EMPNO 변수를 NUMBER(4) 자료형으로 선언하며 7788 값을 할당합니다.

03행 VENAME 변수를 VARCHAR2(10) 자료형으로 선언합니다. VENAME 변수는 선언만 하고 값을 지정하지 않습니다.

04행 BEGIN 실행부를 시작합니다.

05행 선언만 한 VENAME 변수에 문자열 SCOTT를 저장합니다.

06~07행 V\_EMPNO, VENAME 변수 값을 출력합니다.

08행 현재 블록을 종료합니다.

09행 작성한 PL/SQL문을 실행합니다.

### 상수 정의하기

저장한 값이 필요에 따라 변하는 변수와 달리 상수(constant)는 한번 저장한 값이 프로그램이 종료될 때까지 유지되는 저장 요소입니다. 상수를 선언할 때 다음과 같이 기존 변수 선언에 CONSTANT 키워드를 지정합니다.

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

기본 형식

**①** **②** **③** **④** **⑤**

번호	설명
①	데이터를 저장할 변수 이름을 지정합니다. 이 변수 이름을 통해 저장한 데이터를 사용하게 됩니다.
②	선언한 변수를 상수로 정의합니다. 즉 한번 저장한 값은 변하지 않습니다.
③	선언한 변수에 저장할 데이터의 자료형을 지정합니다.
④	선언한 변수에 값을 할당하기 위해 :=를 사용합니다. 이 기호는 오른쪽 값을 왼쪽 변수에 대입하겠다는 뜻입니다. 값을 할당하지 않고 변수를 선언만 한다면 ③, ④는 생략할 수 있습니다.
⑤	변수에 저장할 첫 데이터 값이나 저장할 수 있는 값을 결과로 반환하는 표현식을 지정합니다. 이 값은 변수에 지정한 자료형과 맞아야 합니다.

### 실습 16-5 상수에 값을 대입한 후 출력하기

• 완성 파일 16-5.sql

```
01  DECLARE
02      V_TAX CONSTANT NUMBER(1) := 3;
03  BEGIN
04      DBMS_OUTPUT.PUT_LINE('V_TEX : ' || V_TAX);
05  END;
06  /
```

:: 결과 화면

V\_TEX : 3

상수로 선언한 V\_TAX의 자료형은 NUMBER이기 때문에 숫자 3을 대입할 수 있습니다.

### 변수의 기본값 지정하기

DEFAULT 키워드는 변수에 저장할 기본값을 지정합니다. 다음 ③, ④를 사용하여 변수의 기본값을 명시합니다.

변수 이름 자료형 **DEFAULT** 값 또는 값 도출되는 여러 표현식;  
\_\_\_\_\_

기본 형식

①

②

③

④

번호	설명
①	데이터를 저장할 변수 이름을 지정합니다. 이 변수 이름을 통해 저장한 데이터를 사용하게 됩니다.
②	선언한 변수에 저장할 데이터의 자료형을 지정합니다.
③	DEFAULT 키워드를 작성하여 변수의 기본값을 명시합니다.
④	변수에 저장할 첫 데이터 값이나 저장할 수 있는 값을 결과로 반환하는 표현식을 지정합니다. 이 값은 변수에 지정한 자료형과 맞아야 합니다.

### 실습 16-6 변수에 기본값을 설정한 후 출력하기

• 완성 파일 16-6.sql

```
01  DECLARE
02      V_DEPTNO NUMBER(2) DEFAULT 10;
03  BEGIN
04      DBMS_OUTPUT.PUT_LINE('V_DEPTNO : ' || V_DEPTNO);
05  END;
06  /
```

:: 결과 화면

V\_DEPTNO : 10

변수 V\_DEPTNO의 기본값은  
10으로 설정되었습니다.

## 변수에 NULL 값 저장 막기

특정 변수에 NULL이 저장되지 않게 하려면 NOT NULL 키워드를 사용합니다. PL/SQL에서 선언한 변수는 특정 값을 할당하지 않으면 NULL값이 기본으로 할당됩니다. 이러한 이유로 NOT NULL 키워드를 사용한 변수는 반드시 선언과 동시에 특정 값을 지정해 주어야 합니다.

변수 이름 자료형 NOT NULL := 또는 DEFAULT 값 또는 값이 도출되는 여러 표현식;					기본 형식
번호	1	2	3	4	5

번호	설명
①	데이터를 저장할 변수 이름을 지정합니다. 이 변수 이름을 통해 저장한 데이터를 사용하게 됩니다.
②	선언한 변수에 저장할 데이터의 자료형을 지정합니다.
③	NOT NULL 키워드를 작성하여 변수에 NULL이 저장되지 못하도록 막습니다.
④, ⑤	변수에 저장할 첫 데이터 값이나 저장할 수 있는 값을 결과로 반환하는 표현식을 지정합니다. 이 값은 변수에 지정한 자료형과 맞아야 합니다. 그리고 NULL이 아닌 값을 할당해야 합니다. DEFAULT 키워드를 함께 사용하여 변수의 기본값으로 설정할 수도 있습니다.

:=를 사용하여 선언과 동시에 값을 할당하고 DEFAULT 키워드를 함께 사용하여 변수의 기본값으로 설정할 수도 있습니다.

### 실습 16-7 변수에 NOT NULL을 설정하고 값을 대입한 후 출력하기

· 완성 파일 16-7.sql

```
01  DECLARE
02      V_DEPTNO NUMBER(2) NOT NULL := 10;
03  BEGIN
04      DBMS_OUTPUT.PUT_LINE('V_DEPTNO : ' || V_DEPTNO);
05  END;
06  /
```

:: 결과 화면

V\_DEPTNO : 10

실습 16-7에서 사용하고 있는 변수 V\_DEPTNO의 자료형은 NUMBER이고 변수 값으로 NULL을 가질 수 없습니다.

```

01  DECLARE
02      V_DEPTNO NUMBER(2) NOT NULL DEFAULT 10;
03  BEGIN
04      DBMS_OUTPUT.PUT_LINE('V_DEPTNO : ' || V_DEPTNO);
05  END;
06  /

```

:: 결과 화면

V\_DEPTNO : 10

변수 V\_DEPTNO는 NULL을 가질  
수 없고 기본값은 10입니다.

### 변수 이름 정하기

변수를 포함한 PL/SQL문에서 지정하는 객체 이름을 식별자(identifier)라고 합니다. 식별자에 이름을 붙이는 규칙은 다음과 같습니다.

1. 같은 블록 안에서 식별자는 고유해야 하며 중복될 수 없습니다.
2. 대·소문자를 구별하지 않습니다.
3. 테이블 이름 붙이는 규칙과 같은 규칙을 따릅니다.
  - ① 이름은 문자로 시작해야 합니다(한글도 가능하며 숫자로 시작할 수 없음).
  - ② 이름은 30byte 이하여야 합니다(영어는 30자, 한글은 15자까지 사용 가능).
  - ③ 이름은 영문자(한글 가능), 숫자(0-9), 특수문자(\$, #, \_)를 사용할 수 있습니다.
  - ④ SQL 키워드는 테이블 이름으로 사용할 수 없습니다(SELECT, FROM 등은 테이블 이름으로 사용 불가).

### 변수의 자료형

변수에 저장할 데이터가 어떤 종류인지를 특정 짓기 위해 사용하는 자료형은 크게 스칼라(scalar), 복합(composite), 참조(reference), LOB(Large OBject)로 구분됩니다. 여기에서는 스칼라형과 참조형에 대해 알아보겠습니다.

#### 스칼라형

스칼라형(scalar type)은 숫자, 문자열, 날짜 등과 같이 오라클에서 기본으로 정의해 놓은 자료형으로 내부 구성 요소가 없는 단일 값을 의미합니다.

☞ 스칼라형은 C, Java 등의 프로그래밍 언어에서 원시(primitive) 자료형과 유사합니다.

스칼라형은 숫자·문자열·날짜·논리 데이터로 나뉘며 대표적인 스칼라형은 다음과 같습니다.

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

② PL/SQL에 사용하는 자료형의 좀 더 자세한 내용은 오라클 공식 문서(docs.oracle.com/cd/B28359\_01/app-dev.111/b28370/datatypes.htm#i46029)를 참조하세요.

앞에서 변수 선언 및 값을 대입하기 위해 작성하고 실행한 예제에 사용한 데이터는 모두 스칼라 자료형입니다.

### 참조형

참조형(reference type)은 오라클 데이터베이스에 존재하는 특정 테이블 열의 자료형이나 하나의 행 구조를 참조하는 자료형입니다. 열을 참조할 때 %TYPE, 행을 참조할 때 %ROWTYPE을 사용합니다. %TYPE의 사용법은 다음과 같습니다. %TYPE으로 선언한 변수는 지정한 테이블 열과 완전히 같은 자료형이 됩니다.

변수 이름	테이블이름.열이름%TYPE;	기본 형식
①	②	③

번호	설명
①	데이터가 저장될 변수의 이름을 지정합니다. 이 변수 이름을 통해 저장된 데이터를 사용하게 됩니다.
②	특정 테이블에 속한 열의 이름을 명시합니다. ① 변수는 명시된 테이블의 열과 같은 크기의 자료형이 지정됩니다.
③	앞에서 지정한 테이블의 열과 같은 자료형 및 크기임을 명시합니다. 이후 := 또는 DEFAULT 키워드를 사용하여 값을 먼저 지정해 줄 수도 있습니다.

#### 실습 16-9 참조형(열)의 변수에 값을 대입한 후 출력하기

• 완성 파일 16-9.sql

```

01  DECLARE
02      V_DEPTNO DEPT.DEPTNO%TYPE := 50;
03  BEGIN
04      DBMS_OUTPUT.PUT_LINE('V_DEPTNO : ' || V_DEPTNO);
05  END;
06  /

```

:: 결과 화면

V\_DEPTNO : 50

특정 테이블에서 하나의 열이 아닌 행 구조 전체를 참조할 때 %ROWTYPE을 사용합니다.

변수 이름 테이블 이름%ROWTYPE;  
①      ②      ③

기본 형식

번호	설명
①	데이터를 저장할 변수 이름을 지정합니다. 이 변수 이름을 통해 저장한 데이터를 사용하게 됩니다.
②	특정 테이블을 지정합니다. ① 번수는 명시된 테이블 결과 같은 종류의 데이터를 가지게 됩니다.
③	%TYPE과는 달리 저장할 값을 직접 지정할 수 없습니다.

다음은 %ROWTYPE을 활용하여 변수를 선언하고 있습니다. V\_DEPTNO\_ROW 변수가 DEPT 테이블의 행 구조를 참조하도록 선언되었습니다. 즉 V\_DEPTNO\_ROW 변수는 내부에 DEPTNO, DNAME, LOC 필드를 가지게 됩니다.

#### 실습 16-10 참조형(행)의 변수에 값을 대입한 후 출력하기

• 완성 파일 16-10.sql

```
01  DECLARE
02      V_DEPT_ROW DEPT%ROWTYPE;
03  BEGIN
04      SELECT DEPTNO, DNAME, LOC INTO V_DEPT_ROW
05      FROM DEPT
06      WHERE DEPTNO = 40;
07      DBMS_OUTPUT.PUT_LINE('DEPTNO : ' || V_DEPT_ROW.DEPTNO);
08      DBMS_OUTPUT.PUT_LINE('DNAME : ' || V_DEPT_ROW.DNAME);
09      DBMS_OUTPUT.PUT_LINE('LOC : ' || V_DEPT_ROW.LOC);
10  END;
11 /
```

:: 결과 화면

DEPTNO : 40

DNAME : OPERATIONS

LOC : BOSTON

02행 V\_DEPT\_ROW 변수를 DEPT 테이블의 행 구조로 선언합니다. 이제 V\_DEPT\_ROW 변수는 내부에 DEPTNO, DNAME, LOC 필드를 가지게 됩니다.

04행 INTO V\_DEPT\_ROW를 사용하여 DEPT 테이블의 SELECT문 결과 행을 V\_DEPT\_ROW에 대입합니다. INSERT문과 마찬가지로 V\_DEPT\_ROW가 소유한 필드 개수 및 자료형과 SELECT문 결과 열의 개수와 자료형은 같아야 합니다.

### 복합형, LOB형

스칼라형과 참조형 외에도 PL/SQL에서는 복합형(composite type)과 LOB형을 사용할 수 있습니다. 이 중 복합형은 여러 종류 및 개수의 데이터를 저장하기 위해 사용자가 직접 정의하는 자료형으로 컬렉션(collection), 레코드(record)로 구분되며 이 내용은 17장에서 다룹니다.

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 있습니다.

☞ 복합형과 LOB형에 대해 더 많은 내용을 알고 싶다면 다음 오리를 문서를 확인해 주세요.

복합형 : [docs.oracle.com/cd/B28359\\_01/appdev.111/b28370/collections.htm#LNPLS005](http://docs.oracle.com/cd/B28359_01/appdev.111/b28370/collections.htm#LNPLS005)

LOB형 : [docs.oracle.com/cd/B28359\\_01/appdev.111/b28370/datatypes.htm#CIHJABDI](http://docs.oracle.com/cd/B28359_01/appdev.111/b28370/datatypes.htm#CIHJABDI)

1분  
복습

다음 빈칸을 채우며 복습해 보세요.

PL/SQL에서 많이 사용하는 자료형은 숫자·문자열·날짜 등과 같이 오라클 데이터베이스에서 기본으로 정의해 둔 단일 값을 저장하는 <sup>1</sup> 스 이 있습니다. <sup>2</sup> 참 은 오라클 데이터베이스에 존재하는 특정 테이블 열의 자료형이나 하나의 행 구조를 참고하는 자료형입니다.

정답 1.숫자타입 2.참조타입

## 16-3 조건 제어문

특정 조건식을 통해 상황에 따라 실행할 내용을 달리하는 방식의 명령어를 조건문이라고 합니다. PL/SQL에서는 IF문과 CASE문을 사용할 수 있습니다.

### IF 조건문

PL/SQL에서 제공하는 IF 조건문은 다음과 같이 세 가지 방식을 사용할 수 있습니다.

종류	설명
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

여러 프로그래밍 언어에서 사용하는 단일 IF문과 같은 역할을 하는 IF-THEN문은 다음과 같이 사용합니다. 주어진 조건식의 결과 값이 true인 경우에는 작업을 수행하지만, false 또는 NULL일 경우에는 작업을 수행하지 않고 다음 내용을 실행합니다.

IF 조건식 THEN -①  
수행할 명령어; -②  
END IF; -③

기본 형식

번호	설명
①	true 또는 false 판별이 가능한 조건식을 지정합니다. 여러 연산자 및 함수를 사용할 수 있습니다.
②	조건식의 결과 값이 true일 때 실행할 명령어를 지정합니다. 여러 명령어 지정이 가능합니다
③	IF를 종료합니다.

### 실습 16-11 | 변수에 입력한 값이 홀수인지 알아보기(입력 값이 홀수일 때)

· 완성 파일 16-11.sql

```
01  DECLARE
02      V_NUMBER NUMBER := 13;
03  BEGIN
04      IF MOD(V_NUMBER, 2) = 1 THEN
05          DBMS_OUTPUT.PUT_LINE('V_NUMBER는 홀수입니다!');
06  END IF;
07  END;
08 /
```

:: 결과 화면

V\_NUMBER는 홀수입니다!

02행 V\_NUMBER 변수에 13을 대입합니다.

04행 MOD 함수를 사용하여 V\_NUMBER 값을 2로 나눈 나머지 값이 1인지 아닌지 검사합니다(V\_NUMBER에는 13이 저장되어 있으므로 결과 값은 true입니다).

05행 조건식의 결과 값이 true일 때 화면에 메시지를 출력합니다.

06행 IF문을 끝냅니다.

만약 위 예제에서 V\_NUMBER 변수에 짝수인 14를 대입하면 실습 16-12와 같이 화면에 출력되는 내용이 없습니다. 짝수를 2로 나눈 나머지는 0이 되어 조건식의 결과 값이 false가 되기 때문이죠.

### 실습 16-12 | 변수에 입력된 값이 홀수인지 알아보기(입력 값이 짝수일 때)

· 완성 파일 16-12.sql

```
01  DECLARE
02      V_NUMBER NUMBER := 14;
03  BEGIN
04      IF MOD(V_NUMBER, 2) = 1 THEN
05          DBMS_OUTPUT.PUT_LINE('V_NUMBER는 홀수입니다!'); 05행은 IF 조건문의 결과가
06  END IF;                                         true일 때만 실행됩니다.
07  END;
08 /
```

## IF-THEN-ELSE

IF-THEN-ELSE문은 지정한 조건식의 결과 값이 true일 경우에 실행할 명령어와 조건식의 결과 값이 true가 아닐 때 실행할 명령어를 각각 지정할 수 있습니다.

```

IF 조건식 THEN -①
  수행할 명령어; -②
ELSE
  수행할 명령어; -③
END IF; -④

```

기본 형식

번호	설명
①	true 또는 false 판별이 가능한 조건식을 지정합니다. 여러 연산자 및 함수 사용이 가능합니다.
②	조건식의 결과 값이 true일 경우 실행할 명령어를 지정합니다. 여러 명령어 지정이 가능합니다.
③	조건식의 결과 값이 true가 아닐 경우 실행할 명령어를 지정합니다. 여러 명령어 지정이 가능합니다.
④	IF를 종료합니다.

다음 PL/SQL문을 실행해 봅시다. V\_NUMBER 변수에 짝수인 14를 대입하고 V\_NUMBER 변수 값을 2로 나누면 나머지는 0이 되기 때문에 조건식 결과는 false가 됩니다. 즉 ELSE문에 지정된 명령어가 실행됩니다. IF-THEN-ELSE문은 특정 조건식의 만족 여부에 따른 두 가지 상황에 각각 수행할 작업 내용을 지정할 수 있습니다.

#### 실습 16-13     변수에 입력된 값이 홀수인지 짝수인지 알아보기(입력 값이 짝수일 때)     • 완성 파일 16-13.sql

```

01  DECLARE
02      V_NUMBER NUMBER := 14;
03  BEGIN
04      IF MOD(V_NUMBER, 2) = 1 THEN
05          DBMS_OUTPUT.PUT_LINE('V_NUMBER는 홀수입니다!');
06      ELSE
07          DBMS_OUTPUT.PUT_LINE('V_NUMBER는 짝수입니다!');
08      END IF;
09  END;
10  /

```

:: 결과 화면

V\_NUMBER는 짝수입니다!

02행 V\_NUMBER 변수에 14를 대입합니다.

04행 MOD 함수를 사용하여 V\_NUMBER 값을 2로 나눈 나머지 값이 1인지 아닌지 검사합니다(V\_NUMBER에는 14가 저장되어 있으므로 결과 값은 false입니다).

06~07행 IF문에 지정한 조건식의 결과 값이 true가 아닐 경우에 수행할 명령어를 지정합니다.

### IF-THEN-ELSIF

결과 값이 true인지 아닌지 여부에 따라 두 가지 상황을 구현할 수 있는 IF-THEN-ELSE문과 달리 IF-THEN-ELSIF문은 여러 종류의 조건을 지정하여 각 조건을 만족하는 경우마다 다른 작업의 수행을 지정하는 것이 가능합니다.



번호	설명
① ~ ②	① 조건식의 결과 값이 true이면 ② 명령어를 수행하고(다음 ELSIF 및 ELSE문은 실행되지 않음) 조건식의 결과 값이 false면 다음 ELSIF 조건식으로 넘어갑니다.
③ ~ ⑥	③ 조건식의 결과 값이 true이면 ④ 명령어를 수행하고(다음 ELSIF 및 ELSE문은 실행되지 않음) 조건식의 결과 값이 false이면 다음 ELSIF 또는 ELSE로 넘어갑니다. ⑤, ⑥ 과정도 같습니다.
⑦ ~ ⑧	⑦ 위 IF, ELSIF 조건식의 결과 값이 어디에서도 true가 나오지 않는다면 ⑧ 명령어를 수행합니다.
⑨	IF를 종료합니다.

다음 PL/SQL문은 점수에 따라 학점을 주기 위해 여러 조건식을 지정합니다. 위에서부터 다음으로 조건식을 하나씩 실행하여 true가 되는 영역의 작업이 수행될 것입니다. 점수가 60점이 되지 않는다면 모든 조건식을 지나치고 ELSE문의 명령어가 실행되어 F학점이 나옵니다.

실습 16-14      입력한 점수가 어느 학점인지 출력하기(IF-THEN-ELSIF 사용)      · 완성 파일 16-14.sql

```
01  DECLARE
02      V_SCORE NUMBER := 87;
03  BEGIN
04      IF V_SCORE >= 90 THEN
05          DBMS_OUTPUT.PUT_LINE('A학점');
```

```

06    ELSIF V_SCORE >= 80 THEN
07        DBMS_OUTPUT.PUT_LINE('B학점');
08    ELSIF V_SCORE >= 70 THEN
09        DBMS_OUTPUT.PUT_LINE('C학점');
10    ELSIF V_SCORE >= 60 THEN
11        DBMS_OUTPUT.PUT_LINE('D학점');
12    ELSE
13        DBMS_OUTPUT.PUT_LINE('F학점');
14    END IF;
15 END;
16 /

```

:: 결과 화면

B학점

## CASE 조건문

CASE 조건문도 IF조건문과 마찬가지로 조건식의 결과 값에 따라 여러 가지 수행 작업을 지정할 수 있습니다. IF-THEN-ELSIF문과 같이 조건식의 결과 값이 여러 가지일 때 CASE 조건문을 좀 더 단순하게 표현할 수 있습니다. CASE 조건문은 다음과 같이 두 가지 방식을 사용합니다.

종류	설명
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

단순 CASE문은 비교 기준(여러 가지 결과 값이 나올 수 있는)이 되는 변수 또는 식을 명시합니다. 그리고 각 결과 값에 따라 수행할 작업을 지정합니다.

```

CASE 비교 기준 -①
WHEN 값1 THEN -②
    수행할 명령어; -③
WHEN 값2 THEN -④
    수행할 명령어; -⑤
...
ELSE -⑥
    수행할 명령어; -⑦
END CASE; -⑧

```

기본 형식

번호	설명
①	① 먼저 여러 결과 값이 나올 수 있는 비교 기준을 지정합니다. 변수 또는 여러 표현식을 지정할 수 있습니다.
② ~ ⑤	② 비교 기준의 결과 값이 값1과 일치하면 ③ 작업을 수행하고 나머지 명령어를 건너뜁니다. ④, ⑤ 역시 같습니다.
⑥ ~ ⑦	⑥ 위의 WHEN에 일치하는 값을 찾지 못한다면 마지막으로 ELSE에 지정한 ⑦ 작업을 수행합니다.
⑧	CASE문을 끝냅니다.

앞의 IF-THEN-ELSIF문에서 사용한 예제와 마찬가지로 학점을 출력하는 조건문을 CASE 조건문으로 구현해 볼까요? TRUNC(V\_SCORE/10)을 비교 기준으로 V\_SCORE 변수 값은 10으로 나눈 후 소수점을 버린 결과 값에 따라 학점을 출력할 수 있습니다.

#### 실습 16-15    입력 점수에 따른 학점 출력하기(단순 CASE 사용)

• 완성 파일 16-15.sql

```

01  DECLARE
02      V_SCORE NUMBER := 87;
03  BEGIN
04      CASE TRUNC(V_SCORE/10)
05          WHEN 10 THEN DBMS_OUTPUT.PUT_LINE('A학점');
06          WHEN 9 THEN DBMS_OUTPUT.PUT_LINE('B학점');
07          WHEN 8 THEN DBMS_OUTPUT.PUT_LINE('C학점');
08          WHEN 7 THEN DBMS_OUTPUT.PUT_LINE('D학점');
09          WHEN 6 THEN DBMS_OUTPUT.PUT_LINE('F학점');
10      ELSE DBMS_OUTPUT.PUT_LINE('F학점');
11  END CASE;
12 END;
13 /

```

:: 결과 확인

B학점

#### 검색 CASE

검색 CASE문은 비교 기준을 명시하지 않고 각각의 WHEN 절에서 조건식을 명시한 후 해당 조건을 만족할 때 수행할 작업을 정해 줍니다.

```

CASE
  WHEN 조건식1 THEN -①
    수행할 명령어; -②
  WHEN 조건식2 THEN -③
    수행할 명령어; -④
  ...
  ELSE -⑤
    수행할 명령어; -⑥
END CASE;

```

기본 형식

번호	설명
① ~ ④	① 조건식1의 결과 값이 true라면 ② 작업을 수행합니다. 나머지 명령어는 건너뜁니다. ③, ④로 표 현한 다른 WHEN절 역시 동일하게 동작합니다.
⑤ ~ ⑥	⑤ 먼저 실행한 WHEN절의 조건식을 만족하는 경우가 없다면 ⑥ 작업을 수행합니다.

실습 16-15와 마찬가지로 점수에 따른 학점을 출력하기 위해 다음과 같이 검색 CASE문을 사용할 수 있습니다. 이렇게 각 조건을 WHEN절에 명시하면 사실 IF-THEN-ELSIF와 큰 차이가 없습니다.

#### 실습 16-16 입력 점수에 따른 학점 출력하기(검색 CASE 사용)

• 완성 파일 16-16.sql

```

01  DECLARE
02    V_SCORE NUMBER := 87;
03  BEGIN
04    CASE
05      WHEN V_SCORE >= 90 THEN DBMS_OUTPUT.PUT_LINE('A학점');
06      WHEN V_SCORE >= 80 THEN DBMS_OUTPUT.PUT_LINE('B학점');
07      WHEN V_SCORE >= 70 THEN DBMS_OUTPUT.PUT_LINE('C학점');
08      WHEN V_SCORE >= 60 THEN DBMS_OUTPUT.PUT_LINE('D학점');
09      ELSE DBMS_OUTPUT.PUT_LINE('F학점');
10    END CASE;
11  END;
12  /

```

:: 결과 확인  
B학점

06장 오라클 함수에서 언급한 CASE문과 PL/SQL문의 CASE문은 사용 방법이 유사합니다. 하지만 SQL문에 사용하는 CASE문이 조건에 따라 특정 결과 값을 반환하는 것에 그치는 데 반해 PL/SQL문의 CASE 조건문은 조건에 따라 수행할 작업을 지정할 수 있다는 차이가 있습니다. SQL문의 CASE는 END로 종료되며 PL/SQL문의 CASE 조건문은 END CASE로 종료 된다는 점도 기억해 주세요.

## 16-4 반복 제어문

반복문은 특정 작업을 반복하여 수행하고자 할 때 사용합니다. PL/SQL에서는 다음 네 가지 반복문을 제공하고 있습니다. 여기에서는 기본 LOOP, WHILE LOOP, FOR LOOP 반복문을 소개하겠습니다.

종류	설명
기본 LOOP	기본 반복문
WHILE LOOP	특정 조건식의 결과를 통해 반복 수행
FOR LOOP	반복 횟수를 정하여 반복 수행
Cursor FOR LOOP	커서를 활용한 반복 수행

위에서 나열한 반복문 외에도 반복 수행을 중단시키거나 특정 반복 주기를 건너뛰는 다음 명령어도 함께 살펴보겠습니다.

종류	설명
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

기본 LOOP문은 매우 간단한 형태의 반복문입니다. 반복을 위한 별다른 조건 없이 반복할 작업 내용을 지정해 줍니다.

```
LOOP  
  반복 수행 작업;  
END LOOP;
```

기본 형식

기본 LOOP문은 반복의 종료 시점이나 조건식을 따로 명시하지 않으므로 지정한 작업을 무한히 반복 수행하게 됩니다. 이러한 현상을 무한 루프(Infinite Loop)라고 합니다. 하지만 수행 작업을 무한 반복해야 하는 경우는 많지 않으므로 대부분의 기본 LOOP문은 반복을 종료하는 EXIT 명령어를 함께 사용합니다.

그러면 기본 LOOP문을 활용한 실습 16-17을 실행해 볼까요? V\_NUM 변수는 기본 LOOP문의 반복이 수행됨에 따라 값이 1씩 증가하며 EXIT-WHEN문을 사용하여 값이 4보다 클 때 반복을 종료하도록 구현합니다.

### 실습 16-17 기본 LOOP 사용하기

• 완성 파일 16-17.sql

```
01  DECLARE
02      V_NUM NUMBER := 0;
03  BEGIN
04      LOOP
05          DBMS_OUTPUT.PUT_LINE('현재 V_NUM : ' || V_NUM);
06          V_NUM := V_NUM + 1;
07          EXIT WHEN V_NUM > 4;
08      END LOOP;
09  END;
10  /
```



07행이 바로 반복 수행을 종료하는 조건문입니다.

:: 결과 화면

```
현재 V_NUM : 0
현재 V_NUM : 1
현재 V_NUM : 2
현재 V_NUM : 3
현재 V_NUM : 4
```

만약 EXIT-WHEN 대신 EXIT 문을 사용한다면 IF 조건문을 함께 사용하여 오른쪽과 같이 구현할 수 있습니다.

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

## WHILE LOOP

WHILE LOOP문은 반복 수행 여부를 결정하는 조건식을 먼저 지정한 후 조건식의 결과 값이 true일 때 조건을 반복하고 false가 되면 반복을 끝냅니다.

**WHILE** 조건식 **LOOP**

기본 형식

반복 수행 작업;

**END LOOP;**

실습 16-17에서 0부터 4까지 변수 값을 증가시킨 기본 LOOP문과 같은 작업을 수행하는 WHILE LOOP문을 제작해 보죠. 앞의 예제에서 EXIT-WHEN문에 명시한 조건과는 반대의 조건식 V\_NUM < 4를 사용합니다. EXIT-WHEN문은 조건식의 결과 값이 true일 때 반복을 끝낸다는 뜻이므로 WHILE문은 조건식의 결과 값이 true가 될 때까지 반복을 수행하겠다는 의미입니다. 비슷하지만 정반대로 반복 여부를 결정하는 것입니다.

### 실습 16-18 WHILE LOOP 사용하기

· 완성 파일 16-18.sql

```
01  DECLARE
02      V_NUM NUMBER := 0;
03  BEGIN
04      WHILE V_NUM < 4 LOOP
05          DBMS_OUTPUT.PUT_LINE('현재 V_NUM : ' || V_NUM);
06          V_NUM := V_NUM + 1;
07      END LOOP;
08  END;
09 /
```

:: 결과 화면

현재 V\_NUM : 0

현재 V\_NUM : 1

현재 V\_NUM : 2

현재 V\_NUM : 3

WHILE LOOP문은 반복 수행이 시작되기 전에 조건식을 검사하므로 조건식의 결과 값에 따라 단 한 번도 반복 수행되지 않을 수도 있음을 기억하세요.

## FOR LOOP

FOR LOOP문은 반복의 횟수를 지정할 수 있는 반복문으로 다음과 같은 형태로 작성합니다. 지정한 시작 값부터 1씩 증가하여 종료 값에 이를 때까지 작업을 반복 수행합니다. FOR 키워드 다음에 작성한 i는 반복 수행 중의 시작 값과 종료 값 사이의 현재 숫자가 저장되는 특수한 변수로 카운터(counter)라고 합니다. 카운터는 선언부에서 정의하지 않고 FOR LOOP문에서 바로 정의하여 사용합니다. FOR LOOP문 안에서만 사용 가능하며, 값을 임의로 할당할 수 없고 현재 저장되어 있는 값을 참조만 할 수 있습니다.

```
FOR i IN 시작 값 .. 종료 값 LOOP  
    반복 수행 작업;  
END LOOP;
```

기본 형식

다음 FOR LOOP문은 0부터 4까지 총 다섯 번의 반복을 수행하는 반복문입니다. 반복이 수행됨에 따라 카운터 값이 증가하고 있는 것을 확인해 주세요.

### 실습 16-19 WHILE LOOP 사용하기

• 완성 파일 16-19.sql

```
01 BEGIN  
02     FOR i IN 0..4 LOOP  
03         DBMS_OUTPUT.PUT_LINE('현재 i의 값 : ' || i);  
04     END LOOP;  
05 END;  
06 /
```

:: 결과 화면

```
현재 i의 값 : 0  
현재 i의 값 : 1  
현재 i의 값 : 2  
현재 i의 값 : 3  
현재 i의 값 : 4
```

시작 값에서 종료 값을 역순으로 반복하고 싶다면 다음과 같이 REVERSE 키워드를 사용합니다. 하지만 FOR LOOP문에 사용하는 시작 값과 종료 값의 위치는 변하지 않으므로 주의하세요.

```
FOR i IN REVERSE 시작 값 .. 종료 값 LOOP  
    반복 수행 작업;  
END LOOP;
```

기본 형식

### 실습 16-20 FOR LOOP 사용하기

· 완성 파일 16-20.sql

```
01 BEGIN
02   FOR i IN REVERSE 0..4 LOOP
03     DBMS_OUTPUT.PUT_LINE('현재 i의 값 : ' || i);
04   END LOOP;
05 END;
06 /
```

:: 결과 화면

현재 i의 값 : 4

현재 i의 값 : 3

현재 i의 값 : 2

현재 i의 값 : 1

현재 i의 값 : 0

REVERSE 키워드를 사용하여 역순으로 반복 수행을 지정할 때에도 시작 값과 종료 값의 지정 위치는 그대로입니다.

## CONTINUE문, CONTINUE-WHEN문

CONTINUE문과 CONTINUE-WHEN문은 오라클 11g 버전부터 사용할 수 있습니다. 반복 수행 중 CONTINUE가 실행되면 현재 반복 주기에 수행해야 할 남은 작업을 건너뛰고 다음 반복 주기로 바로 넘어가는 효과가 있습니다. EXIT, EXIT-WHEN문과 마찬가지로 CONTINUE는 즉시 다음 반복 주기로 넘어가고, CONTINUE-WHEN문은 특정 조건식을 만족할 때 다음 반복 주기로 넘어가게 됩니다.

앞의 FOR LOOP문에서 사용한 예제에 CONTINUE-WHEN문을 추가해 보죠. 실습16-21 을 살펴보면 조건식에는 MOD(i, 2) = 1을 지정했습니다. 현재 FOR LOOP문의 카운터 값을 2로 나눈 나머지 값이 1인 경우, 즉 카운터가 홀수이면 남은 명령문을 수행하지 않고 다음 반복 주기로 넘어가겠다는 뜻입니다. 카운터가 짝수일 때만 출력된 것을 확인해 주세요. CONTINUE문을 잘 활용하면 반복문의 분기 처리를 한층 간편하게 사용할 수 있습니다.

### 실습 16-21 FOR LOOP 안에 CONTINUE문 사용하기

· 완성 파일 16-21.sql

```
01 BEGIN
02   FOR i IN 0..4 LOOP
03     CONTINUE WHEN MOD(i, 2) = 1;
04     DBMS_OUTPUT.PUT_LINE('현재 i의 값 : ' || i);
05   END LOOP;
06 END;
07 /
```

:: 결과 화면

현재 i의 값 : 0

현재 i의 값 : 2

현재 i의 값 : 4

조건 제어문과 반복 제어문 외에도 GOTO문과 NULL문 같은 순차 제어문(sequential control)도 있습니다. PL/SQL 프로그래밍에서 크게 중요한 부분은 아니므로 이 책에서는 소개를 생략합니다. 좀 더 자세한 조건 제어문과 반복 제어문, 순차 제어문의 정보가 필요하다면 오라클 공식 문서([docs.oracle.com/cd/B28359\\_01/appdev.111/b28370/controlstructures.htm#LNPLS004](http://docs.oracle.com/cd/B28359_01/appdev.111/b28370/controlstructures.htm#LNPLS004))와 관련 자료를 참고하세요.

● 이 장에서 배운 내용을 실습하며 정리하세요.

- Q1** 숫자 1부터 10까지의 숫자 중 오른쪽과 같이 훌수만 출력하는 PL/SQL 프로그램을 작성해 보세요.

```
SQL> SET SERVEROUTPUT ON;
SQL> BEGIN
... [PL/SQL 작성]
/
현재 i의 값 : 1
현재 i의 값 : 3
현재 i의 값 : 5
현재 i의 값 : 7
현재 i의 값 : 9
PL/SQL 처리가 정상적으로 완료되었습니다.
SQL>
```

- Q2** DEPT 테이블의 DEPTNO와 자료형이 같은 변수 V\_DEPTNO를 선언합니다. 그리고 V\_DEPTNO 변수 값에 10, 20, 30, 40을 대입했을 때 다음과 같이 부서 이름을 출력하는 프로그램을 작성해 보세요. 단 부서 번호가 10, 20, 30, 40이 아니면 N/A로 출력합니다.

```
SQL> SET SERVEROUTPUT ON;
SQL> DECLARE
... [변수 선언 및 값 대입]
BEGIN
... [PL/SQL 작성]
/
DNAME : ACCOUNTING
PL/SQL 처리가 정상적으로 완료되었습니다.
SQL>
```

V\_DEPTNO 변수에 10을 대입했을 때

```
SQL> SET SERVEROUTPUT ON;
SQL> DECLARE
... [변수 선언 및 값 대입]
BEGIN
... [PL/SQL 작성]
/
DNAME : N/A
PL/SQL 처리가 정상적으로 완료되었습니다.
SQL>
```

V\_DEPTNO 변수에 10, 20, 30, 40 이외의 값을 대입했을 때

정답 이지스피클리싱 홈페이지에서 확인하세요.