**1. 변수**

SQL과 절차적인 문장 안에서 PL/SQL로써 변수를 선언할 수 있고 그것을 사용할 수 있다.

**1.1 변수 사용**

변수는 자료를 일시적으로 저장하고 변경하고 검증하기 위해 하나 또는 그 이상의 변수를 선언하여 사용한다. 또한 변수는 데이터베이스를 액세스하지 않고 계산이나 다른 데이터 조작에 사용할 수 있다. 이러한 변수는 일단 선언되면 다른 선언적 문장을 포함한 다른 문장에서 간단하게 그것을 반복적으로 참조하여 사용할 수 있다.

%TYPE와 %ROWTYPE을 사용하여 변수를 선언하면 테이블의 구조가 변경(데이터형과 길이)되어도 Application에서는 실행 시간에 테이블을 참조하여 변수가 정의되므로 데이터의 독립성, 유지비용 절감을 제공하고, 새로운 업무 요구에 충족시키기 위해 데이터베이스 변경에 따라 프로그램의 적응, 수정되는 것을 허용한다.

**1.2 PL/SQL에서 변수 처리**

1. 선언 섹션 내에서 변수를 선언하고 초기화하여 사용합니다.
2. 실행 섹션에서 변수에 대한 새 값을 할당 합니다.
3. 매개변수를 통해 PL/SQL 블록으로 값을 전달합니다.
4. 출력 변수를 통해 결과를 봅니다.

문제1) EMP 테이블에 EMPNO\_SEQUENCE의 SEQUENCE를 이용하여 이름.급여,부서번호를 입력받아 등록하는 SCRIPT를 작성하여라. 단 10번부서는 입력된 급여에 20%의 가산하여 등록하고 30번부서는 10% 가산 점이 있다.

|  |
| --- |
| SET VERIFY OFF  ACCEPT p\_name PROMPT ' 이 름: '  ACCEPT p\_sal PROMPT ' 급 여: '  ACCEPT p\_deptno PROMPT ' 부서번호: '  DECLARE  v\_name VARCHAR2(10) := UPPER('&p\_name');  v\_sal NUMBER(7,2) := &p\_sal;  v\_deptno NUMBER(2) := &p\_deptno;  BEGIN  IF v\_deptno = 10 THEN  v\_sal := v\_sal \* 1.2;  ELSIF v\_deptno = 30 THEN  v\_sal := v\_sal \* 1.1;  END IF;  INSERT INTO emp(empno,ename,sal,deptno)  VALUES (empno\_sequence.NEXTVAL,v\_name,v\_sal,v\_deptno);  COMMIT;  END;  /  SET VERIFY ON |

**1.3 변수 유형**

모든 PL/SQL 변수는 저장 포맷,제약 조건, 값의 유효 범위를 지정하는 데이터형을 가지고 있다. PL/SQL은 변수, 상수, 포인터를 선언하기 위해 사용할 수 있는 4가지 데이터형(Scalar, Composite, Reference, LOB(large objects:Oracle8)을 지원합니다.

**1.3.1 PL/SQL 변수**

1. Scalar : 주로 단일 값을 보유합니다. 주요 데이터형은 ORACLE SERVER 테이블의 열 유형에 대응하는 것들입니다.
2. Composite : 레코드 같은 조합 데이터형은 PL/SQL 블록에서 조작되고 정의되는 필드 그룹을 허용합니다.
3. Reference : 참조 데이터형은 pointer라 불리며 다른 프로그램 항목을 지시하는 값을 보유합니다.
4. LOB(large objects) : LOB 데이터형은 locator라 불리며 라인 밖에서 지정된 큰 객체의 위치를 지정하는 값을 보유합니다.

**1.3.2 Non-PL/SQL 변수**

1. Bind와 host variables

**1.4 PL/SQL 변수 선언**

PL/SQL 블록에서 그것을 참조하기 전에 선언 섹션에서 모두 PL/SQL 식별자를 선언할 필요가 있습니다. 초기값을 할당하기 위해 옵션을 가집니다. 변수를 선언하기 위해 변수에 대한 값을 할당할 필요는 없습니다. 선언에서 다름 변수를 참조한다면 이전 문장에서 개별적으로 그것들을 반드시 선언해 놓아야 합니다.

**1.4.1 Syntax**

|  |
| --- |
| identifier [**CONSTANT**] datatype [**NOT NULL**]  [:= | **DEFAULT** expression]; |

identifier 변수의 이름

CONSTANT 변수의 값을 변경할 수 없도록 제약합니다.

datatype Scalar, Composite, Reference, LOB(large objects)

NOT NULL 값을 포함해야만 하도로 하기 위해 변수를 제약 합니다.

Expression Literal, 다른 변수, 연산자나 함수를 포함하는 표현식

**1.4.2 사용 예**

|  |
| --- |
| DECLARE  v\_hiredate DATE;  v\_deptno NUMBER(2) NOT NULL := 10;  v\_loc VARCHAR2(13) := ‘ATLANTA’;  v\_com CONSTANT NUMBER := 1400; |

☞ Guidelines

1. SQL 객체에 대해 사용된 동일한 이름 지정 규칙에 따라 식별자의 이름을 지정한다.
2. 이름 지정 규약을 사용할 수 있습니다.(예:v\_name는 변수를 나타내고 c\_name는 상수를 나타낸다)
3. NOT NULL로 지정된 변수를 초기화 합니다.
4. 지정 연산자(:=)를 사용하거나 예약어 DEFAULT를 사용하여 식별자를 초기화 합니다.
5. 한 라인에 하나의 식별자만 선언 합니다.
6. 상수 선언에서 CONSTRAINT는 형 지정자보다 먼저 기술되어야 한다.

**1.5 이름 지정 규칙**

1. 하나의 블록에서 동일 이름의 변수를 선언할 수 없습니다.
2. 블록이 다르면 동일 이름을 선언할 수 있습니다. 객체들이 동시에 존재하는 곳에서는 현재 블록에서 정의된 객체들만이 사용될 수 있습니다.
3. 변수에 대한 이름을 블록에서 사용되는 테이블 열의 이름과 동일하게 선택해서는 안됩니다. PL/SQL 변수가 SQL 명령에서 사용되고 열과 동일 이름을 가지면 ORACLE SERVER은 참조되는 열로 간주 합니다.

**1.6 변수의 값 지정**

변수의 값을 지정하거나 재지정하기 위해 PL/SQL 지정 문자를 사용합니다. 지정 연산자(:=)의 좌측에 새 값을 받기 위한 변수를 적습니다.

**1.6.1 Syntax**

|  |
| --- |
| identifier := expression; |

**1.6.2 사용 예**

|  |
| --- |
| DECLARE  v\_hiredate DATE;  v\_ename VARCHAR2(10);  BEGIN  v\_hiredate := ‘30-DEC-98’;  v\_ename := ‘MADURO’; |

♣ 주의

Oracle Server에서 DEFAULT 날짜 형식 설정이 데이터베이스마다 차이가 있을 수 있기 때문에 DEFAULT 날짜 형식을 알아야 한다. DEFAULT 날짜 형식이 YY-MM-DD이면 v\_hiredate := ‘99-01-01’로 값을 설정하여야 한다. 일반적으로 날짜의 형식에 의존하지 않고 사용하고자 할 경우에는 v\_hiredate := TO\_DATE(‘99-01-01’,’YY-MM-DD’);을 사용한다.

**1.7 스칼라 데이터 형**

1. 단일 값을 유지 합니다.
2. 내부적인 구성 요소는 없습니다.

**1.7.1 기본 스칼라 데이터 형**

|  |  |
| --- | --- |
| 데이터 형 | 설 명 |
| VARCHAR2(n) | 변수 길이 문자 데이터에 데한 기본형은 32767Byte까지 입니다. VARCHAR2변수와 상수에 대한 디폴트 크기는 없습니다. |
| NUMBER(p,s) | 고정(fixed)과 유동(floating)포인트 숫자에 대한 기본형 |
| DATE | 날짜와 시간에 대한 기본형. DATE값은 지정 이후의 초 단위로 날에 대한 시간을 포함합니다. 날짜의 범위는 BC 4712년 1월1일부터 AD 9999년 12월 31일사이 입니다. |
| CHAR(n) | 고정 길이 문자에 대한 기본형은 32767바이트까지 입니다. 지정하지 않는다면 디폴트 길이는 1로 설정됩니다. |
| LONG | 고정 길이 문자에 대한 기본형은 32760바이트까지 입니다.  LONG 데이터베이스 열의 최대 폭은 2147483647바이트입니다. |
| LONG RAW | 이진 데이터와 바이트 문자열에 대한 기본형은 32760Byte까지 입니다. LONG RAW 데이터는 PL/SQL에 의해 해석되지 않습니다. |
| BOOLEAN | 계산에 사용되는 3가지 가능한 값 가운데 기본형(TRUE,FALSE,NULL) |
| BINARY\_INTEGER | -2147483647~2147483647사이의 정수에 대한 기본 형 |
| PLS\_INTEGER | -2147483647~2147483647사이의 signed정수에 대한 기본형으로  PLS\_INTEGER 값은 NUMBER와 BINARY\_INTEGER값보다 적은 기억장치를 필요로 합니다. |

♣ 참고

LONG 데이터 형은 LONG값의 최대 길이가 32767 바이트인 것을 제외하고는 VARCHAR2와 유사합니다. 그러므로 32760바이트보다 더 긴 값은 LONG 데이터베이스 열에서 LONG PL/SQL 변수로 사용할 수 없습니다.

**1.7.2 스칼라 변수 선언의 예**

|  |
| --- |
| DECLARE  v\_job VARCHAR2(9);  v\_count BINARY\_INTEGER := 0;  v\_total\_sal NUMBER(9,7) := 0;  v\_order\_date DATE := SYSDATE + 7;  v\_tax\_rate CONSTANT NUMBER(3,2) := 8.25;  v\_valid BOOLEAN NOT NULL := TRUE;  v\_sex CHAR(1); |

**1.7.3 %TYPE 속성**

변수의 데이터 형과 정밀도를 직접 코딩하기 보다는 이전에 선언된 다른 변수 또는 데이터베이스 열에 맞추어 변수를 선언하기 위해 %TYPE속성을 사용할 수 있습니다. 변수에 저장되는 값이 데이터베이스의 테이블에서 오거나 변수가 테이블에 쓰여지기로 되었다면 %TYPE 속성은 자주 사용됩니다. 변수 선언에서 필요한 데이터형 대신에 속성을 사용하려면 데이터베이스 테이블과 열 이름을 접두어로 사용합니다. 또한 이전에 선언된 변수를 참조한다면 속성 앞에 변수명을 기술합니다. 데이터베이스 수준에서 테이블의 데이터형을 변경하여도 PL/SQL을 고칠 필요가 없습니다.

**가) 사용 예**

|  |
| --- |
| DECLARE  v\_empno emp.empno%TYPE;  v\_ename emp.ename%TYPE;  v\_deptno emp.deptno%TYPE := 10; |

**1.7.4 BOOLEAN 변수 선언**

1. TRUE, FALSE, NULL값만을 BOOLEAN 변수에 대해 지정할 수 있습니다.
2. 변수는 논리연산자 AND, OR, NOT에 의해 접속 됩니다.
3. 변수는 항상 TRUE, FALSE, NULL을 생성 합니다.
4. 산술, 문자, 날짜 표현식은 BOOLEAN값을 리턴하기 위해 사용될 수 있습니다.

**가) 사용 예**

|  |
| --- |
| DECLARE  v\_sal1 NUMBER(5) := 5000;  v\_sal2 NUMBER(5) := 6000;  v\_flag BOOLEAN := TRUE;  BEGIN  v\_flag := (v\_sal1 > v\_sal2);  IF v\_flag THEN  . . . . . . . . . . |

**1.8 조합 데이터 형(Composite Datatype)**

Composite Datatype은 내부 구성 요소를 갖고 있고 PL/SQL에서 사용할 수 있는 Composite Datatype은 RECORD,TABLE,중첩 TABLE,VARRAY입니다. RECORD 데이터형은 관련은 있으나 서로 다른 데이터형들을 논리적인 하나의 단위로 묶기 위해 사용하고 TABLE 데이터형은 전체 객체로써 데이터형이 같은 데이터의 모음을 참조하고 조작하기 위해 사용한다. 한번 정의되면 테이블과 레코드는 재이용할 수 있다.

**1.8.1 PL/SQL TABLE TYPE**

테이블형의 객체는 PL/SQL 테이블이라 불립니다. PL/SQL 테이블은 행에 대해 배열처럼 액세스하기 위해 기본키를 사용합니다. 배열과 유사하고 PL/SQL 테이블을 액세스하기 위해 BINARY\_INTEGER데이터형의 기본키와 PL/SQL 테이블 요소를 저장하는 스칼라 또는 레코드 데이터형의 열을 포함하여야 한다. 또한 이들은 동적으로 자유롭게 증가할 수 있습니다.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Primary key |  | Column |  |
|  | . . . . . . |  | . . . . . . |  |
|  | 1 |  | Jones |  |
|  | 2 |  | Smith |  |
|  | 3 |  | Maduro |  |
|  | . . . . . . |  | . . . . . . |  |
|  | BINARY\_INTEGER |  | 스칼라 |  |

**가) Syntax**

|  |
| --- |
| **TYPE** table\_type\_name **IS TABLE OF**  {column\_type | variable%**TYPE** | table.column%**TYPE**} [NOT NULL]  [**INDEX BY BINARY\_INTEGER**];  identifier table\_type\_name; |

table\_type\_name 테이블형의 이름

column\_type VARCHAR2,DATE,NUMBER과 같은 스칼라 데이터 형

identifier 전체 PL/SQL 테이블을 나타내는 식별자의 이름

문제2) TABLE 변수를 사용하여 EMP 테이블에서 이름과 업무를 출력하여라.

|  |
| --- |
| SET SERVEROUTPUT ON  DECLARE  TYPE ename\_table\_type IS TABLE OF emp.ename%TYPE  INDEX BY BINARY\_INTEGER;  TYPE job\_table\_type IS TABLE OF emp.job%TYPE  INDEX BY BINARY\_INTEGER;  ename\_table ename\_table\_type;  job\_table job\_table\_type;  i BINARY\_INTEGER := 0;  BEGIN  FOR k IN (SELECT ename,job FROM emp) LOOP  i := i + 1;  ename\_table(i) := k.ename;  job\_table(i) := k.job;  END LOOP;  FOR j IN 1..i LOOP  DBMS\_OUTPUT.PUT\_LINE(RPAD(ename\_table(j),12) ||  RPAD(job\_table(j),9));  END LOOP;  END;  /  SET SERVEROUTPUT OFF |

**1.8.2 PL/SQL RECORD TYPE**

PL/SQL RECORD TYPE은 데이터베이스의 테이블 ROW와 다르고 3GL에서의 RECORD나 STRUCTURE와 유사하다. PL/SQL RECORD는 Scalar, PL/SQL RECORD, PL/SQL TABLE 데이터 타입 중 하나 이상의 요소를 갖고 있어야 하며, 다른 데이터 타입을 가질 수도 있다. 또한 FIELD(ITEM)들의 집합을 하나의 논리적 단위로 처리할 수 있게 해 주므로 테이블의 ROW를 읽어올 때 편리하다.

**가) Syntax**

|  |
| --- |
| **TYPE** type\_name **IS RECORD**  (field\_name1 {scalar\_datatype|record\_type} [**NOT NULL**] [{:= | **DEFAULT**} expr],  (field\_name2 {scalar\_datatype|record\_type} [**NOT NULL**] [{:= | **DEFAULT**} expr],  . . . . . . .);  identifiee\_name type\_name; |

type\_name RECODE 형의 이름, 이 식별자는 RECODE를 선언하기 위해 사용한다.

field\_name RECODE내의 필드명

**나) RECORD 참조**

RECORD에서 필드는 이름으로 액세스 됩니다. 개별 필드를 참조하거나 초기화 하기 위해 “.”을 사용합니다.

|  |
| --- |
| Record\_name.field\_name |

**다) RECORD에 대한 값 할당**

SELECT 또는 FETCH문장을 사용함으로써 RECORD에 값을 지정할 수 있다. 열이름의 RECORD의 필드와 동일한 순서로 1대1대응을 하여야 합니다. 두 RECORD가 동일한 구조를 가지면 하나의 RECORD를 다른 RECORD에 지정할 수 있습니다.

|  |
| --- |
| Record\_name1.field\_name1 := expression; |
| Record\_name1.field\_name1 := Record\_name2.field\_name2; |

문제3) EMP 테이블에서 이름을 입력 받아 아래의 형태로 출력하는 SCRIPT를 작성하여라.

|  |
| --- |
| 조회하고자 하는 사원의 이름을 입력하시오 : scott  사원번호 : 7788  이 름 : SCOTT  담당업무 : ANALYST  메 니 저 : 7566  입사일자 : 09-DEC-82  급 여 :  보 너 스 : $0  부서번호 : 20 |

|  |
| --- |
| SET VERIFY OFF  SET SERVEROUTPUT ON  ACCEPT p\_ename PROMPT '조회하고자 하는 사원의 이름을 입력하시오 : '  DECLARE  TYPE emp\_record\_type IS RECORD(  v\_empno emp.empno%TYPE,  v\_ename emp.ename%TYPE,  v\_job emp.job%TYPE,  v\_mgr emp.mgr%TYPE,  v\_hiredate emp.hiredate%TYPE,  v\_sal emp.sal%TYPE,  v\_comm emp.comm%TYPE,  v\_deptno emp.deptno%TYPE);  emp\_record emp\_record\_type;  v\_ename emp.ename%TYPE := '&p\_ename';  BEGIN  SELECT \*  INTO emp\_record  FROM emp  WHERE ename = UPPER(v\_ename);  DBMS\_OUTPUT.PUT\_LINE('사원번호 : ' || TO\_CHAR(emp\_record.v\_empno));  DBMS\_OUTPUT.PUT\_LINE('이 름 : ' || emp\_record.v\_ename);  DBMS\_OUTPUT.PUT\_LINE('담당업무 : ' || emp\_record.v\_job);  DBMS\_OUTPUT.PUT\_LINE('메 니 저 : ' || TO\_CHAR(emp\_record.v\_mgr));  DBMS\_OUTPUT.PUT\_LINE('입사일자 : ' || TO\_CHAR(emp\_record.v\_hiredate));  DBMS\_OUTPUT.PUT\_LINE('급 여 : ' ||  LTRIM(TO\_CHAR(emp\_record.v\_sal,'$999,990.00')));  DBMS\_OUTPUT.PUT\_LINE('보 너 스 : ' ||  LTRIM(TO\_CHAR(NVL(emp\_record.v\_comm,0),'$999,990')));  DBMS\_OUTPUT.PUT\_LINE('부서번호 : ' || TO\_CHAR(emp\_record.v\_deptno));  EXCEPTION  WHEN NO\_DATA\_FOUND THEN  DBMS\_OUTPUT.PUT\_LINE('&p\_ename' || '의 자료는 없습니다.');  WHEN TOO\_MANY\_ROWS THEN  DBMS\_OUTPUT.PUT\_LINE('&p\_ename' || '자료가 2건 이상입니다.');  WHEN OTHERS THEN  DBMS\_OUTPUT.PUT\_LINE('기타 에러 입니다.');  END;  /  SET VERIFY ON  SET SERVEROUTPUT OFF |

☞ Guidelines

1. 각 RECORD는 필요한 만큼 많은 필드를 가질 수 있다.
2. RECORD는 초기값을 지정할 수 있고 NOT NULL로 정의될 수 있다.
3. 초기값이 없는 필드는 NULL로 초기화 됩니다.
4. DEFAULT 키워드는 필드를 정의할 때 사용될 수 있습니다.
5. 임의의 블록 서브 프로그램,패키지의 선언 부분에 RECORD형을 정의하고 사용자 정의 RECORD를 선언할 수 있다.
6. 중첩 RECORD를 선언하고 사용할 수 있다. RECORD는 다른 RECORD의 구성 요소가 될 수 있다.

**1.8.3 %ROWTYPE의 속성**

데이터베이스의 테이블 또는 VIEW의 일련의 열을 RECORD로 선언하기 위하여 %ROWTYPE를 사용합니다. 데이터베이스 테이블 이름을 %ROWTYPE 앞에 접두어를 붙여 RECORD를 선언하고 FIELD는 테이블이나 VIEW의 COLUMN명과 데이터 타입과 LENGTH을 그대로 가져올 수 있다.

**가) Syntax**

|  |
| --- |
| identifier reference%**ROWTYPE;** |

identifier RECORD에 대해 지정된 이름

reference RECORD의 기초가 되는 테이블,VIEW,CURSOR,변수 명을 기술

**나) 개별 필드를 참조하는 방법**

|  |
| --- |
| record\_name.field\_name |

**다) %ROWTYPE을 사용 시 장점**

1. 알지 못하는 데이터베이스 COLUMN의 개수와 데이터 형식을 모르게 지정할 수 있다.
2. 실행 시 변경되는 데이터베이스 COLUMN의 개수와 데이터 형식을 지정할 수 있다.
3. SELECT 문장으로 행을 검색할 때 유리하다.

문제4) EMP 테이블에서 이름을 입력 받아 아래의 형태로 출력하는 SCRIPT를 작성하여라.

|  |
| --- |
| 조회하고자 하는 사원의 이름을 입력하시오 : scott  사원번호 : 7788  이 름 : SCOTT  담당업무 : ANALYST  메 니 저 : 7566  입사일자 : 09-DEC-82  급 여 :  보 너 스 : $0  부서번호 : 20 |

|  |
| --- |
| SET VERIFY OFF  SET SERVEROUTPUT ON  ACCEPT p\_ename PROMPT '조회하고자 하는 사원의 이름을 입력하시오 : '  DECLARE  emp\_record emp%ROWTYPE;  v\_ename emp.ename%TYPE := '&p\_ename';  BEGIN  SELECT \*  INTO emp\_record  FROM emp  WHERE ename = UPPER(v\_ename);  DBMS\_OUTPUT.PUT\_LINE('사원번호 : ' || TO\_CHAR(emp\_record.empno));  DBMS\_OUTPUT.PUT\_LINE('이 름 : ' || emp\_record.ename);  DBMS\_OUTPUT.PUT\_LINE('담당업무 : ' || emp\_record.job);  DBMS\_OUTPUT.PUT\_LINE('메 니 저 : ' || TO\_CHAR(emp\_record.mgr));  DBMS\_OUTPUT.PUT\_LINE('입사일자 : ' || TO\_CHAR(emp\_record.hiredate));  DBMS\_OUTPUT.PUT\_LINE('급 여 : ' ||  LTRIM(TO\_CHAR(emp\_record.sal,'$999,990.00')));  DBMS\_OUTPUT.PUT\_LINE('보 너 스 : ' ||  LTRIM(TO\_CHAR(NVL(emp\_record.comm,0),'$999,990')));  DBMS\_OUTPUT.PUT\_LINE('부서번호 : ' || TO\_CHAR(emp\_record.deptno));  EXCEPTION  WHEN NO\_DATA\_FOUND THEN  DBMS\_OUTPUT.PUT\_LINE('&p\_ename' || '의 자료는 없습니다.');  WHEN TOO\_MANY\_ROWS THEN  DBMS\_OUTPUT.PUT\_LINE('&p\_ename' || '자료가 2건 이상입니다.');  WHEN OTHERS THEN  DBMS\_OUTPUT.PUT\_LINE('기타 에러 입니다.');  END;  /  SET VERIFY ON  SET SERVEROUTPUT OFF |

**1.9 LOB Datatype 변수**

LOB Datatype 변수는 Oracle8 데이터형으로 구조화 되지 않은 데이터(턱스트,그래픽,이미지,비디오 클립,소리 웨이브 폼 같은) 블룩을 4기가 바이트 크기까지 저장할 수 있습니다. LOB Datatype은 데이터에 대한 랜덤 액세스를 지원 합니다.

1. CLOB(character large object) 데이터형은 데이터베이스 내의 단일 바이트 문자 데이터의 대형 블록을 저장하기 위해 사용한다.
2. BLOB(binary large object) 데이터형은 행의 안팎에 데이터베이스 내의 대형 이진 객체를 저장하기 위해 사용됩니다.
3. BFILE(binary file) 데이터형은 데이터베이스 외부의 운영 시스템 파일의 대형 이진 객체를 저장하기 위해 사용됩니다.
4. NCLOB(national language character large object) 데이터형은 데이터베이스 내의 단일 바니트,또는 고정 길이의 멀티바이트NCHAR데이터를 행의 안팎에 저장하기 위해 사용됩니다.

**1.10 바인드 변수**

바인드 변수는 호스트 환경에서 선언된 변수이며, 실행 시간 값을, 그것이 숫자이든지 또는 문자이든지 임의의 다른 변수처럼 그것을 사용할 수 있는 하나 이상의 PL/SQL프로그램의 내부나 외부에서 전달하기 위해 사용합니다. 문장이 프로시저, 함수, 패키지 안에 있지 않는다면, 호출 환경 또는 호스트에서 선언된 변수를 PL/SQL문장에서 참조할 수 있습니다.이것은 선행 컴파일러 프로그램에서 선언된 호스트 언어 변수, Develover/2000Forms어플리케이션의 화면 필드, SQL\*Plus 바인드 변수를 포함합니다.

**1.10.1 바인드 변수 생성**

SQL\*Plus환경에서, 바인드 변수를 선언하기 위해 VARIABLE명령을 사용합니다. SQL과 SQL\*Plus는 바인드 변수를 참조할 수 있고 SQL\*Plus는 그 값을 출력할 수 있습니다.

**가) Syntax**

|  |
| --- |
| **VAR**[**IABLE**] [ variable [ **NUMBER** | **CHAR** (n) | **VARCHAR2** (n)] ] |

**1.10.2 바인드 변수 출력**

SQL\*Plus환경에서 바인드 변수의 현재 값을 출력하기 위해 PRINT명령을 사용합니다.

|  |
| --- |
| SQL> **VARIABLE return\_code NUMBER**  SQL> **DECLARE**  2 **BEGIN**  3 **:return\_code := 100;**  4  **END;**  5 **/**  PL/SQL procedure successfully completed.  SQL> **PRINT return\_code**  RETURN\_CODE  -----------  100 |

**1.11 Non-PL/SQL 변수 참조**

호스트 변수를 참조하기 위해 선언된 PL/SQL변수와 호스트 변수를 구별하기 위해 콜론(:)으로 참조 접두어를 기술하여야 한다.

|  |
| --- |
| :host\_var1 := v\_sal;  :global\_val1 := ‘YES’; |

**2. PL/SQL 블록**

PL/SQL은 프로그램을 논리적인 블록으로 나누게 하는 구조화된 블록 언어이다. PL/SQL 블록에서 사용하는 변수는 블록에 대해 논리적으로 선언할 수 있고 사용할 수 있다. 변수들과 그 외의 식별자를 참조하고 선언함으로써 PL/SQL 블록 내에서 값을 저장하고 변경할 수 있는 논리적인 단위이다.

**2.1 PL/SQL 블록 구문과 지침**

PL/SQL은 SQL의 확장이기 때문에 SQL에 적용하는 일반적인 구문은 PL/SQL 언어에 대해서도 적용 가능하다. 문장은 몇 라인 이상 계속될 수 있습니다. 문법적인 요소는 공백으로 분리될 수 있습니다.

**2.1.1 식별자**

식별자는 상수, 변수, 예외 ,커서,커서 변수,SUBPROGRAM,패키지를 포함하는PL/SQL프로그램 항목과 요소를 명명하기 위해 사용됩니다.

1. 식별자는 30문자까지 포함할 수 있지만 알파벳 문자로 시작해야 합니다.
2. 블록에서 사용된 테이블의 열 이름과 동일한 이름으로 식별자를 사용할 수 없습니다.
3. PL/SQL식별자가 동일 SQL명령에 있고 열로 동일 이름을 가지고 있다면, 이때 ORACLE은 참조 중인 열로 간주합니다.
4. 예약어는 더블 인용 부호에 둘러싸여 있지 않으면 식별자로서 사용될 수 없습니다.
5. 예약어는 읽기 쉽도록 대문자로 사용합니다.

**2.1.2 구분 문자**

구분 문자는 PL/SQL에 대한 특수한 의미를 가지는 단순 상징이거나 혼합 상징 입니다.

|  |  |  |  |
| --- | --- | --- | --- |
| 단순 상징 | | 혼합 상징 | |
| 기호 | 의미 | 기호 | 의미 |
| + | 덧셈 연산자 | <> | 관계형 연산자 |
| - | 뺄셈 연산자 | != | 관계형 연산자 |
| \* | 곱셈 연산자 | || | 접속 연산자 |
| / | 나눗셈 연산자 | -- | 단일 라인 주석 지시자 |
| = | 관계형 연산자 | /\* | 주석 구분 문자 시작 |
| @ | 원격 액세스 지시자 | \*/ | 주석 구분 문자 종료 |
| ; | 문장 종결자 | := | 지정 연산자(치환 연산자) |

**2.1.3 Literal**

Literal은 식별자(identifier)로 표현되지 않은 숫자, 문자, 문자열, BOOLEAN 값입니다.

1. 문자 Literal은 PL/SQL문자 집합 내에서 인쇄 가능한 모든 문자를 포함합니다.
2. 숫자 Literal은 단순 값(예:-32.5) 또는 과학적인 표기법

**2.1.4 주석 코드**

주석 코드 각 단계를 문서화하고 디버깅을 돕기 위해 코드에 주석을 기술합니다. 주석이 단일 라인에 있으면 두 개의 대쉬(--)을 기술하면 – 뒤에 기술된 것은 주석으로 인식 하고, 주석 범위가 여러 줄이라면 기호 /\*와 \*/사이에 주석을 기술 한다. 주석은 철저하게 정보를 제공해야 하고 기능적인 논리 또는 데이터에 대한 어떤 조건 또는 기능을 강요해서는 안됩니다. 좋은 주석은 읽기 쉽게 하고 코드 유지를 위해 매우 중요합니다.

|  |
| --- |
| DECLARE  v\_sal NUMBER(9,2);  BEGIN  /\* Compute the annual salary based on  the monthly salary input from the user. \*/  v\_sal := v\_sal \* 12;  END; -- This is the end of the transaction.  / |

**2.1.5 PL/SQL에서 SQL함수**

1. SQL에서 이용 가능한 대부분의 함수는 PL/SQL표현식에서도 유효합니다.
2. 단일 행 숫자 함수
3. 단일 행 문자 함수
4. 데이터형 변환 함수
5. 데이트 함수 그 밖의 함수
6. 아래 함수는 절차적인 문장에서는 사용 불가능합니다.
7. GREATEST, LEAST, DECODE.
8. 그룹 함수(AVG, MIN, MAX, COUNT, SUM, STDDEVE, VARIANCE)는 테이블에서 행 그룹에 적용되므로 PL/SQL블록에 있는 SQL문장에서만 이용 가능합니다.

|  |
| --- |
| V\_total := SUM(number\_table); -- Error가 발생한다. |

**2.2 데이터형 변환**

PL/SQL은 데이터형이 문장에서 혼합되었다면 동적으로 데이터형 변환을 시도 합니다. 예를 들면NUMBER값을 CHAR 변수가 지정되었다면 그것이 CHAR변수에 저장될 수 있도록 하기 위해 PL/SQL은 동적으로 숫자를 문자로 변환 합니다. 문자 표현을 숫자 값으로 나타내는 역의 상황도 적용됩니다. DATE변수에 대해 문자 값을 지정할 수 있고 역의 상황도 적용됩니다.혼합된 데이터형이 표현식에서 생성되면 데이터를 전환하기 위해 해당 변환 함수를 사용하여야 합니다.

**2.2.1 데이터형 변환 함수의 종류**

1. TO\_CHAR(value, fmt) : value를 문자로 전환
2. TO\_DATE(value, fmt) : value를 날짜 형식으로 전환
3. TO\_NUMBER(value, fmt) : value를 숫자로 전환

☞ Guidelines

PL/SQL은 가능한 한 데이터형 변환을 시도하지만 성공은 수행 중인 작업에 달려있다. 명시적으로 데이터의 형 변환을 시도하는 것은 상당히 성능에 영향을 미치고 소프트웨어 버전이 변경되더라도 유효하게 유지될 수 있기 때문에 좋은 프로그램이 될 수 있다.

**2.3 중첩 블록과 변수 범위**

1. 문장은 실행 명령이 허용하는 곳 어디에서든지 중첩될 수 있습니다.
2. 중첩 블록은 하나의 문장이 됩니다.
3. 예외 섹션에서도 중첩 블록을 사용할 수 있습니다.
4. 객체 범위는 객체를 참조할 수 있는 프로그램 영역 입니다.
5. 선언된 변수를 실행 섹션에서 참조할 수 있습니다.
6. 한정시키지 않은 변수는 참조할 수 있는 영역에서 변수를 참조 가능합니다.
7. 블록은 둘러싸는 블록을 참조할 수 있습니다.
8. 블록은 둘러싸인 블록을 참조할 수 없습니다.

|  |
| --- |
| DECLARE  x NUMBER;  y NUMBER;  BEGIN  x := 1;  y := 2;  -- 이곳에서 x, y의 값은 x := 1, y := 2  DECLARE  y NUMBER;  z NUMBER;  BEGIN  -- 이곳에서 x, y, z의 값은 x := 1, y := NULL, z := NULL  x := 3;  y := 4;  z := 5;  -- 이곳에서 x, y, z의 값은 x := 3, y := 4, z := 5  END;  -- 이곳에서 x, y, z의 값은 x := 3, y := 2, z는 참조 불가능하다.  END;  / |

**2.4 PL/SQL에서 연산자**

연산자는 논리,산술,연결,연산 제어 명령인 괄호,지수 연산자가 있다.

**2.4.1 연산 명령**

표현식에서의 연산은 그것들의 우선 순위에 따라 특별한 순서로 행해 집니다.

|  |  |  |
| --- | --- | --- |
| 우선순위 | 연 산 자 | 설 명 |
| 1 | \*\*, NOT | 지수 승, 논리 부정 연산자 |
| 2 | +, - | 식별, 부정 연산자 |
| 3 | \*, / | 곱셈,나눗셈 연산자 |
| 4 | +, - , || | 덧셈,뺄셈,연결 연산자 |
| 5 | =,!=,<,>,<=,>=,IS NULL,LIKE,BETWEEN,IN | 비교 연산자 |
| 6 | AND | 논리곱 |
| 7 | OR | 논리합 |

**2.4.2 사용 예**

|  |
| --- |
| DECALRE  v\_cnt NUMBER := 0;  v\_eq BOOLEAN;  v\_valid BOOLEAN;  BEGIN  V\_cnt := v\_cnt + 1;  v\_eq := (v\_n1 = v\_n2);  v\_valid := (v\_empno IS NOT NULL);  END;  / |

☞ Guidelines

NULL로 작업할 때 아래 규칙을 명심함으로써 몇 가지 일반적인 실수를 피할 수 있습니다.

1. 관계 연산자로 NULL을 비교하면 항상 NULL이 됩니다.
2. NULL에 논리 연산자 NOT을 적용하면 항상 NULL이 됩니다.
3. 조건 제어 문장에서 조건이 NULL이 되면 관련된 문장은 실행되지 않습니다.

**2.5 프로그래밍 지침 사항**

PL/SQL 블록을 개발할 때 명확한 코드 생성과 유지를 경감하기 위하여 프로그래밍 지침을 수행 합니다.

**2.5.1 코드 규약**

|  |  |  |
| --- | --- | --- |
| 범 주 | 대소문자 규약 | 사용 예 |
| SQL명령어 | 대문자 | SELECT, INSERT, UPDATE |
| PL/SQL 키워드 | 대문자 | DECLARE, BEGIN, END |
| 데이터형 | 대문자 | NUMBER,VARCHAR2,CHAR |
| 식별자와 매개변수 | 소문자 | v\_sal,v\_ename,v\_job |
| 데이터베이스 테이블과 열명 | 소문자 | emp, dept, salgrade |

**2.5.2 코드명 지정 규약**

|  |  |  |
| --- | --- | --- |
| 식별자 | 명명 규약 | 사용 예 |
| 변수 | v\_name | v\_sal |
| 상수 | c\_name | c\_company\_name |
| 커서 | name\_cursur | emp\_cursor |
| 예외 | e\_name | e\_too\_many |
| 테이블 형 | name\_table\_type | amount\_table\_type |
| 테이블 | name\_table | order\_total\_table |
| 레코드 형 | name\_record\_type | emp\_record\_type |
| 레코드 | name\_record | customer\_record |
| SQL\*Plus 치환 매개변수 | p\_name | p\_sal |
| SQL\*Plus 전역변수 | g\_name | g\_year\_sal |

**2.5.3 코드 들여쓰기**

명확성을 위해 또는 읽기 쉽도록 하기 위해 코드를 각 단계별로 들여 씁니다. 구조를 보여주기 위해 Carriage return을 사용합니다.

|  |
| --- |
| DECLARE  v\_deptno emp.deptno%TYPE;  BEGIN  SELECT deptno  INTO v\_deptno  FROM emp  WHERE empno = 7788;  IF v\_deptno IN (10,20) THEN  UPDATE emp  SET sal = 3500  WHERE empno = 7788;  ELSE  UPDATE emp  SET sal = 2500  WHERE empno = 7788;  END IF;  END;  / |

**◈ 연 습 문 제 ◈**

1. %TYPE와 %ROWTYPE를 사용시 장점을 설명하여라.

2. PL/SQL에서 사용할 수 없는 데이터형이 있는가. 있다면 어떤 데이터형인가?

3. 아래의 선언문들을 평가하여라.

|  |
| --- |
| DECLARE  v\_id NUMBER; |
| DECLARE  v\_emp,v\_sal,v\_comm NUMBER; |
| DECLARE  v\_flag BOOLEAN := 1; |
| DECLARE  TYPE emp\_table\_type IS TABLE OF VARCHAR2(20)  INDEX BY BINARY\_INTEGER;  emp\_tabe emp\_table\_type; |

4. 아래와 같은 RECORD TYPE의 변수를 선언하여라.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| v\_empno | v\_ename | v\_job | v\_sal | v\_comm |
| NUMBER(4) | VARCHAR2(10) | VARCHAR2(9) | NUMBER(7,2) | NUMBER(7,2) |

5. PL/SQL에서 사용하는 연산자의 종류를 설명하시오.

6. NULL값이 연산에 참여하면 결과는 어떻게 되는가 ?

7. PL/SQL에서는 들여쓰기를 권장한다 그 이유를 설명하여라.