

Oracle PL SQL



1. **PL/SQL 개요**
2. PROCEDURE VS FUNCTION
3. PL/SQL Data Type
4. PL/SQL 의 SQL
5. PL/SQL 제어문
6. SQL Cursor
7. Exception Handling
8. Package
9. Trigger

01 PL/SQL 이란?

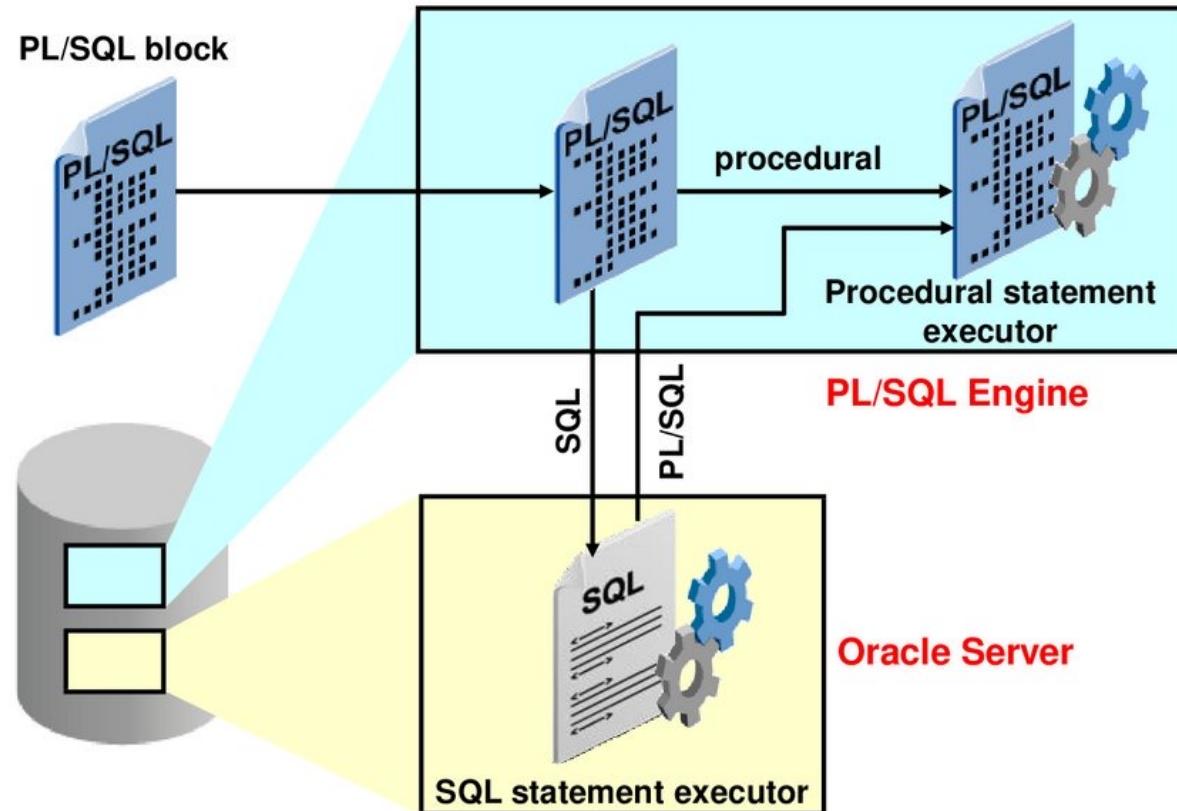
- **PL/SQL** 은 Oracle's Procedural Language extension to SQL 의 약자
- SQL문장에서 **변수정의, 조건처리(IF), 반복처리(LOOP, WHILE, FOR)**등을 지원하며, 오라클 자체에 내장되어 있는 **Procedure Language**
- **DECLARE** 문을 이용하여 정의되며, 선언문의 사용은 선택 사항
- PL/SQL 문은 **블록 구조**로 되어 있고 PL/SQL자신이 컴파일 엔진을 가지고 있다

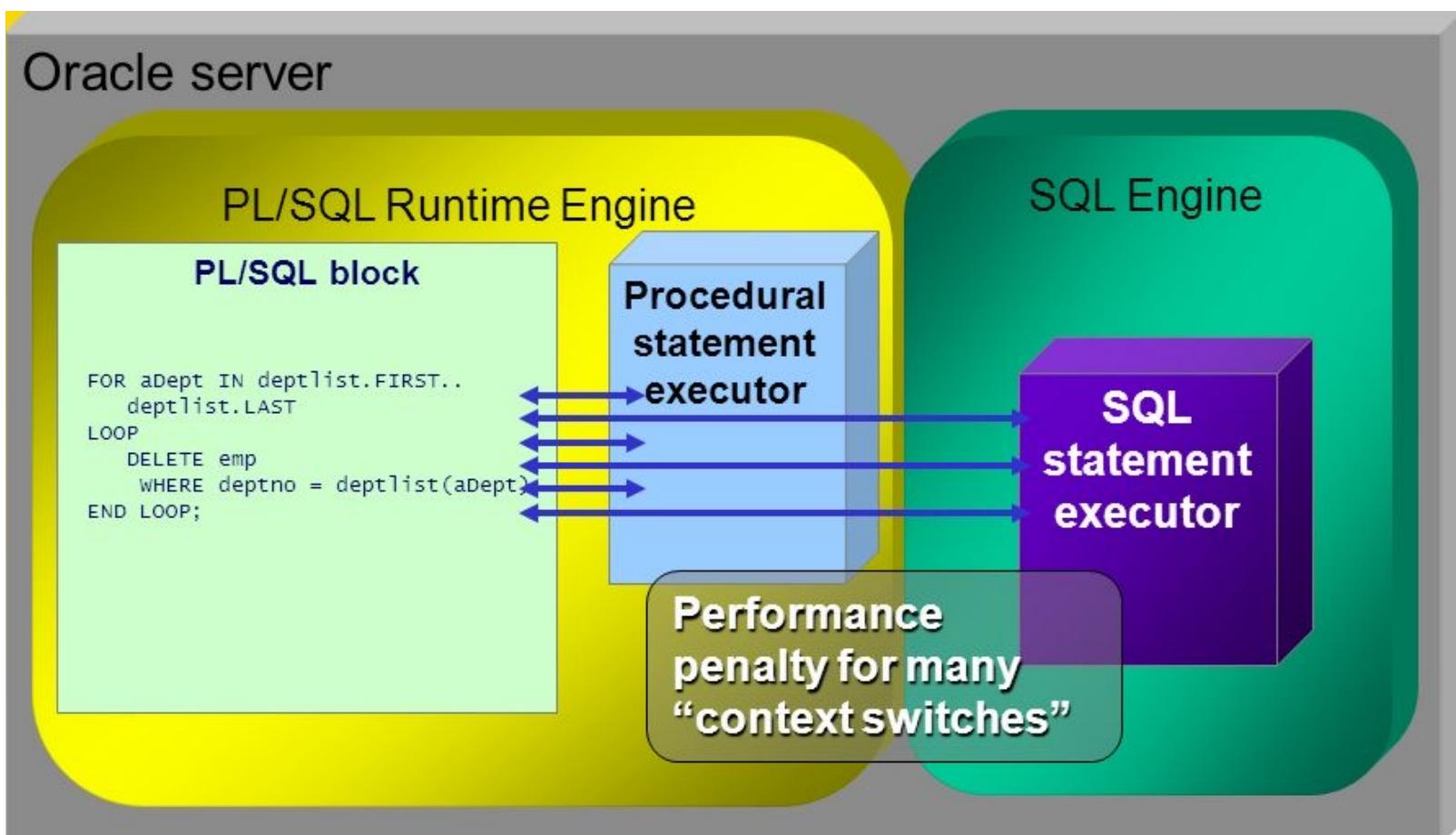
01 PL/SQL 이란?

PL/SQL의 장점

- PL/SQL 문은 BLOCK 구조로 다수의 SQL 문을 한번에 ORACLE DB로 보내서 처리하므로 수행속도를 향상 시킬수 있다
- PL/SQL 의 모든 요소는 하나 또는 두개이상의 블록으로 구성하여 모듈화가 가능
- 보다 강력한 프로그램을 작성하기 위해서 큰 블록안에 소블럭을 위치시킬 수 있다
- **VARIABLE, CONSTANT, CURSOR, EXCEPTION**을 정의하고, SQL문장과 Procedural 문장에서 사용 한다.
- 단순, 복잡한 데이터 형태의 변수를 선언 한다
- 테이블의 데이터 구조와 컬럼명에 준하여 동적으로 변수를 선언 할 수 있다
- **EXCEPTION** 처리 루틴을 이용하여 Oracle Server Error를 처리 한다
- 사용자 정의 에러를 선언하고 **EXCEPTION** 처리 루틴으로 처리 가능 하다

PL/SQL Runtime Architecture





PL/SQL Block Structure

Section	Description
Declarative (optional)	사용할 변수, 상수 선언
Executable (mandatory)	SQL, 제어문, 반복문, 함수정의 등
Exception Handling (optional)	실행중 발생할 수 있는 에러 처리



- PL/SQL은 프로그램을 논리적인 블록으로 나누는 구조화 된 블록 언어
- PL/SQL 블록은 선언부(선택적), 실행부(필수적), 예외 처리부(선택적)로 구성되어 있고, BEGIN과 END 키워드는 반드시 기술해 주어야 한다
- PL/SQL 블록에서 사용하는 변수는 블록에 대해 논리적으로 선언할 수 있고 사용할 수 있다

01 PL/SQL 기본 구조



Declarative Section

- 변수, 상수, CURSOR, USER_DEFINE Exception 선언

Executable Section

- SQL, 반복문, 조건문 실행
- 실행부는 BEGIN으로 시작하고 END로 종료된다.
- 실행문은 프로그램 내용이 들어가는 부분으로서 필수

Exception Handling Section

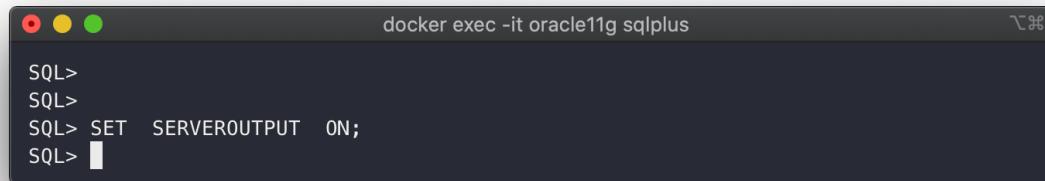
- 예외에 대한 처리
- 일반적으로 오류를 정의하고 처리하는 부분으로 선택 사항

PL/SQL은 기본적으로 처리된 PL/SQL의 결과를 화면에 출력하지 않아 결과를 화면에 출력하기 위해 **화면 출력 활성화**가 필요함

SQL>**SET SERVEROUTPUT ON;**

01 PL/SQL 프로그램 작성 요령

- PL/SQL 블록내에서는 한 문장이 종료할 때마다 세미콜론(:)을 사용 한다
- END뒤에 세미콜론(:)을 사용하여 하나의 블록이 끝났다는 것을 명시 한다
- PL/SQL 블록의 작성은 편집기를 통해 파일로 작성할 수도 있고, SQL 프롬프트에서 바로 작성 할 수도 있다
- SQL*PLUS 환경에서는 DECLARE나 BEGIN이라는 키워드로 PL/SQL블럭이 시작하는 것을 알 수 있다.
- 단일행 주석 : -- 여러행 주석 : /* */
- PL/SQL 블록은 행에 / 가 있으면 종결(실행) 된다
- PL/SQL의 결과를 화면에 출력하기 위해 **화면 출력 활성화가 필요함**



A screenshot of a terminal window titled "docker exec -it oracle11g sqlplus". The window shows the following SQL*Plus session:

```
SQL>
SQL>
SQL> SET SERVEROUTPUT ON;
SQL> 
```

The terminal has a dark background with light-colored text. The title bar is white with blue text. The cursor is at the end of the last command entered.

01 PL/SQL SQL Plus 에서의 한글처리

[윈도우 계열]

시스템 등록정보 -> 고급 -> 환경변수(N)에 다음 추가

NLS_LANG

AMERICAN_AMERICA.KO16KSC5601

or

NLS_LANG

KOREAN_KOREA.KO16KSC5601

[유닉스 계열]

export NLS_LANG=AMERICAN_AMERICA.KO16KSC5601

or

export NLS_LANG=KOREAN_KOREA.AL32UTF8

(계속 등록해보려면 .profile 수정)

select * from nls_database_parameters;

환경 변수

User에 대한 사용자 변수(U)

변수	값
OneDrive	C:\Users\USER\OneDrive
Path	C:\Users\USER\AppData\Local\Program...
PyCharm Commu...	C:\Program Files\JetBrains\PyCharm Co...
TEMP	C:\Users\USER\AppData\Local\Temp
TMP	C:\Users\USER\AppData\Local\Temp

새로 만들기(N)...

편집(E)...

삭제(D)

시스템 변수(S)

변수	값
RTOOLS40_HOME	C:\rttools40
TEMP	C:\WINDOWS\TEMP
TMP	C:\WINDOWS\TEMP
USERNAME	SYSTEM
windir	C:\WINDOWS

새로 만들기(W)...

편집(I)...

삭제(L)

확인

취소

```
docker exec -it oracle11g sqlplus

SQL> select to_char(sysdate, 'YYYY/MM/DD') from dual;

TO_CHAR(SY
-----
2021/03/10

SQL> declare
  2 vno varchar2(20); ←
  3 begin
  4 select to_char(sysdate, 'YYYY/MM/DD') into vno ←
  5 from dual;
  6 dbms_output.put_line(vno); ←
  7 end;
  8 / ←
2021/03/10

PL/SQL procedure successfully completed.

SQL>
```

수행된 결과값 저장할
변수 선언

수행된 결과값을
변수에 저장

변수에 저장된 값을
화면에 출력

PL/SQL 실행

DBeaver 21.0.0 - <Oracle - hr> Script

*<Oracle - hr> Script X DELETE_TEST

```
DECLARE
VNO varchar2(20);
BEGIN
SELECT to_char(sysdate, 'YYYY/MM/DD')
INTO VNO
FROM DUAL;
DBMS_OUTPUT.PUT_LINE(VNO);
END;
```

Output X
2021/03/14

Statistics 1 X

DECLARE VNO varchar2(20); Enter a SQL expression to filter results (use Ctrl+Space)

Name	Value
Updated Rows	-1

Query

```
DECLARE
VNO varchar2(20);
```

Save Cancel Script | 0 row(s) fetched - 2ms

200 1

KST ko_KR 쓰기 가능 스마트 삽입

PL/SQL 실행 결과

PL/SQL 실행
(주의 / 없음)

DBeaver 21.0.0 - <Oracle - hr> Script

*<Oracle - hr> Script X DELETE_TEST

```
DECLARE
VNO varchar2(20);
BEGIN
SELECT to_char(sysdate, 'YYYY/MM/DD')
INTO VNO
FROM DUAL;
DBMS_OUTPUT.PUT_LINE(VNO);
END;|
```

Output X
2021/03/14

Statistics 1 X

DECLARE VNO varchar2(20); Enter a SQL expression to filter results (use Ctrl+Space)

Name	Value
Updated Rows	-1

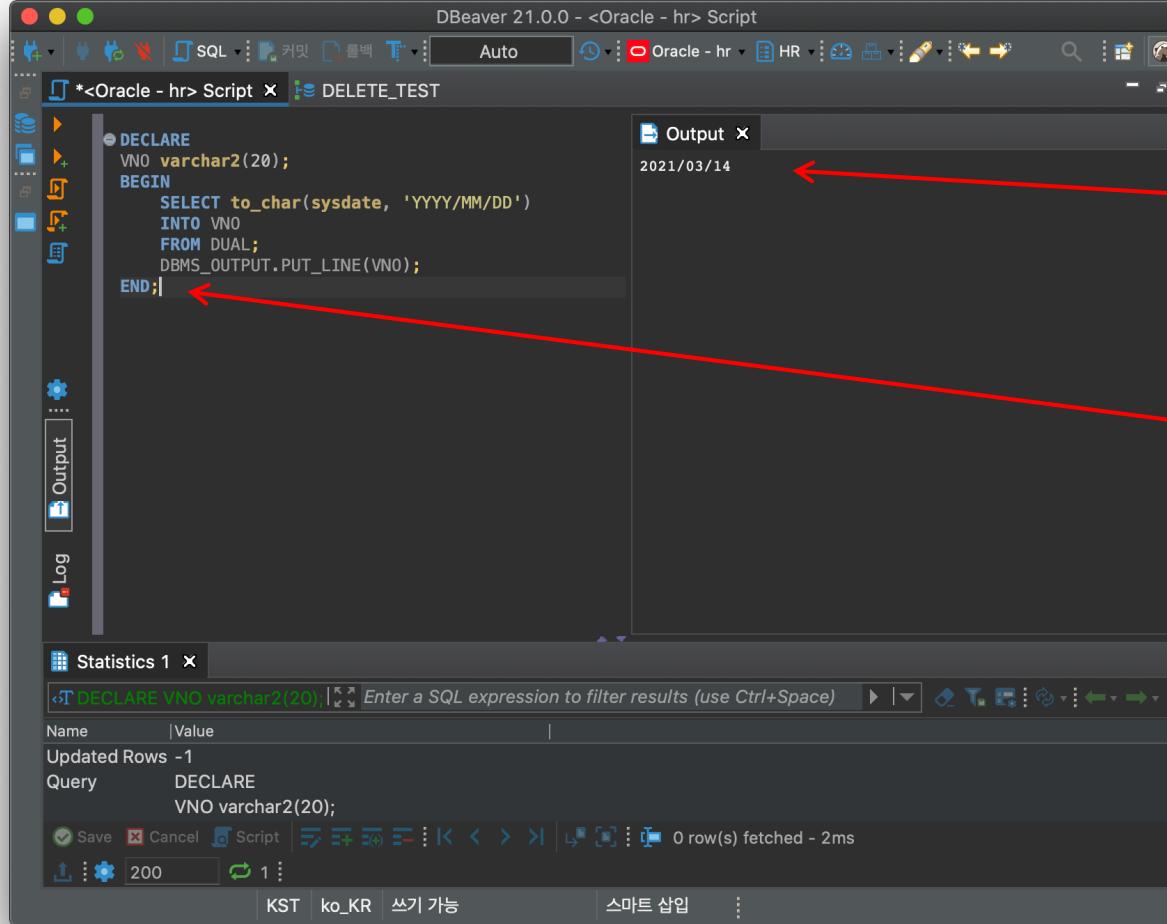
Query

```
DECLARE
VNO varchar2(20);
```

Save Cancel Script | 0 row(s) fetched - 2ms

200 1

KST ko_KR 쓰기 가능 스마트 삽입



PL/SQL 실행 결과

PL/SQL 실행
(주의 / 없음)

```
SELECT select_list
INTO { variable_name [, variable_name] ... | record_name }
FROM table
[WHERE condition]
;
```

```
SQL> declare
  2  vno varchar2(20);
  3  begin
  4    select to_char(sysdate, 'YYYY/MM/DD') into vno
  5    from dual;
  6    dbms_output.put_line(vno);
  7  end;
  8 /
2021/03/10
```

01 PL/SQL 에서 SELECT 사용

연습문제) 사원 테이블을 사용하여 사번을 입력받아 해당 사원의 사번, 이름(first_name), 월급을 출력하세요

```
docker exec -it oracle11g sqlplus
```

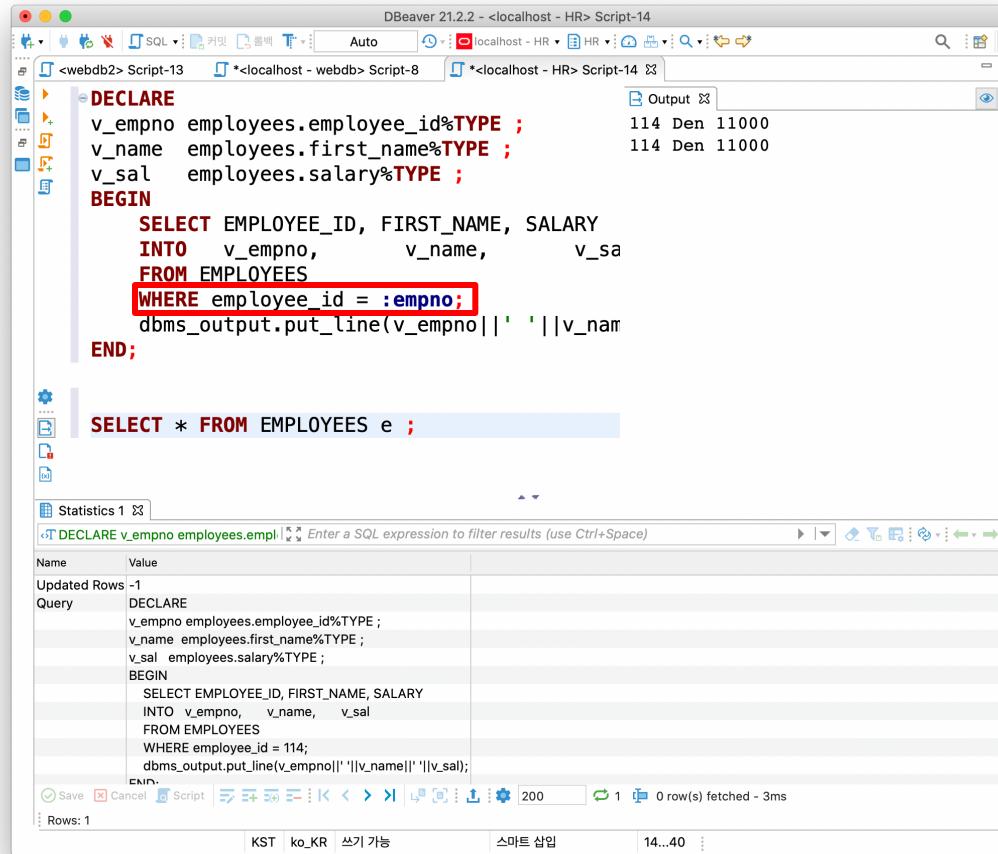
```
SQL> DECLARE
  2   v_empno  employees.employee_id%TYPE ;
  3   v_name    employees.first_name%TYPE ;
  4   v_sal     employees.salary%TYPE;
  5   BEGIN
  6   SELECT employee_id, first_name, salary
  7   INTO v_empno ,v_name, v_sal
  8   FROM employees
  9   WHERE employee_id = '&empno' ;
 10  DBMS_OUTPUT.PUT_LINE(v_empno||' '||v_name||' '||v_sal);
 11  END ;
 12 /
Enter value for empno: 114
old  9: WHERE employee_id = '&empno' ;
new  9: WHERE employee_id = '114' ;
114  Den  12100

PL/SQL procedure successfully completed.

SQL>
```

01 PL/SQL 에서 SELECT 사용

연습문제) 사원 테이블을 사용하여 사번을 입력받아 해당 사원의 사번, 이름(first_name), 월급을 출력하세요



The screenshot shows the DBBeaver interface with the following details:

- Top Bar:** DBeaver 21.2.2 - <localhost - HR> Script-14
- Toolbars:** SQL, 커밋, 롤백, Auto, localhost - HR, HR, Output.
- Script Editor:**

```

DECLARE
    v_empno employees.employee_id%TYPE ;
    v_name employees.first_name%TYPE ;
    v_sal employees.salary%TYPE ;
BEGIN
    SELECT EMPLOYEE_ID, FIRST_NAME, SALARY
    INTO   v_empno,      v_name,      v_sa
    FROM EMPLOYEES
    WHERE employee_id = :empno;
    dbms_output.put_line(v_empno||' '||v_name||v_sal);
END;

SELECT * FROM EMPLOYEES e ;

```
- Statistics View:** Statistics 1

Name	Value
Updated Rows	-1
Query	<pre> DECLARE v_empno employees.employee_id%TYPE ; v_name employees.first_name%TYPE ; v_sal employees.salary%TYPE ; BEGIN SELECT EMPLOYEE_ID, FIRST_NAME, SALARY INTO v_empno, v_name, v_sa FROM EMPLOYEES WHERE employee_id = 114; dbms_output.put_line(v_empno ' ' v_name v_sal); END; </pre>
- Bottom Status Bar:** KST | ko_KR | 쓰기 가능 | 스마트 삽입 | 14...40 | ...

01 PL/SQL 에서 SELECT 사용

연습문제) 사원 테이블을 사용하여 사번을 입력받아 해당 사원의 사번, 이름(first_name), 월급을 출력하세요

The screenshot shows the Oracle SQL Developer interface. In the top navigation bar, the 'hr' schema is selected. The main window displays a PL/SQL script in the '워크시트' tab:

```
1 declare
2   v_empno employees.employee_id%TYPE;
3   v_name  employees.first_name%TYPE;
4   v_sal   employees.salary%TYPE;
5 begin
6   select employee_id, first_name, salary
7     into v_empno,      v_name,      v_sal
8    from employees
9   where employee_id = '&empno';
10  dbms_output.put_line(v_empno||' '||v_name||' '||v_sal);
11 end;
```

A modal dialog titled '대체 변수 입력' (Substitution Variable Input) is displayed, prompting for the value of the substitution variable '&empno'. The input field contains the placeholder 'empno에 대한 값 입력:'.

At the bottom of the screen, there is a '스크립트 출력' (Script Output) tab showing the beginning of another script block:

```
end;
신규:declare
v_empno employees.employee_id%TYPE;
v_name  employees.first_name%TYPE;
v_sal   employees.salary%TYPE;
```

01 PL/SQL 에서 SELECT 사용

연습문제) 사원 테이블을 사용하여 사번을 입력받아 해당 사원의 사번, 이름(first_name), 월급, 입사일을 출력하세요

```
DECLARE
v_empno employees.employee_id%TYPE ;
v_name  employees.first_name%TYPE ;
v_sal   employees.salary%TYPE ;
v_hire_date employees.hire_date%TYPE ;
BEGIN
SELECT EMPLOYEE_ID, FIRST_NAME, SALARY, hire_date
INTO v_empno,      v_name, v_sal, v_hire_date
FROM EMPLOYEES
WHERE employee_id = :empno;
dbms_output.put_line(v_empno||' '||v_name||' '||v_sal||' '||v_hire_date);
END;
```

01 PL/SQL 에서 SELECT 사용

연습문제) 두개의 숫자를 입력받아 합계를 출력하세요

```
SQL>SET VERIFY OFF
SQL>SET SERVEROUTPUT ON
SQL>DECLARE
2  v_no1 NUMBER := &no1 ;
3  v_no2 NUMBER := &no2 ;
4  v_sum NUMBER ;
5
6 BEGIN
7  v_sum := v_no1 + v_no2 ;
8  DBMS_OUTPUT.PUT_LINE('첫번째 수:'||v_no1||', 두번째 수 :'||v_no2||' , 합은 :'||v_sum||' 입니다');
9 END ;
10 /
```

Enter value for no1: 20

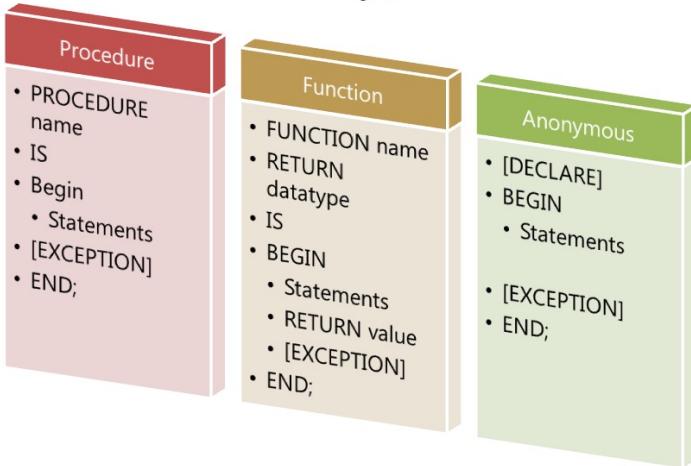
Enter value for no2: 40

첫번째 수: 20, 두번째 수 : 40 , 합은 : 60 입니다

PL/SQL procedure successfully completed.

01 PL/SQL 블럭 타입

Block Types



Anonymous Block (익명 블록)

- 이름이 없는 블록을 의미 하며, 실행하기 위해 프로그램 안에서 선언 되고 실행시에 실행을 위해 PL/SQL 엔진으로 전달 된다
- 선행 컴파일러 프로그램과 SQL*Plus 또는 서버 관리자에서 익명의 블록을 내장 할 수 있다

Procedure (프로시저)

- 특정 작업을 수행할수 있는 이름이 있는 PL/SQL 블록으로서, 매개 변수를 받을수 있고, 반복적으로 사용할수 있다
- 보통 연속 실행 또는 구현이 복잡한 트랜잭션을 수행하는 PL/SQL블록을 데이터베이스에 저장하기 위해 생성 한다

Function (함수)

- 보통 값을 계산하고 결과값을 반환하기 위해서 함수를 많이 사용 한다
- 대부분 구성이 프로시저와 유사하지만 IN 파라미터만 사용 할 수 있고, 반드시 반환 될 값의 데이터 타입을 RETURN문에 선언해야 한다
- 또한 PL/SQL블록 내에서 RETURN문을 통해서 반드시 값을 반환 해야 한다

Oracle PL SQL



1. PL/SQL 개요
2. **PROCEDURE VS FUNCTION**
3. PL/SQL Data Type
4. PL/SQL 의 SQL
5. PL/SQL 제어문
6. SQL Cursor
7. Exception Handling
8. Package
9. Trigger

02 프로시저(Procedure)란?

- 특정 작업을 수행 하는, 이름이 있는 PL/SQL BLOCK 이다
- 매개 변수를 받을 수 있고, 반복적으로 사용 할 수 있는 BLOCK 이다
- 보통 연속 실행 또는 구현이 복잡한 트랜잭션을 수행하는 PL/SQL BLOCK을 데이터베이스에 저장하기 위해 생성 한다

02 프로시저 문법

CREATE OR REPLACE procedure name

IN argument

OUT argument

IN OUT argument

IS

[변수의 선언]

BEGIN --> 필수

[PL/SQL Block]

-- SQL문장, PL/SQL제어 문장

[EXCEPTION] --> 선택

-- error가 발생할 때 수행하는 문장

END; --> 필수

- CREATE OR REPLACE 구문을 사용하여 생성 한다.
- IS 로 PL/SQL의 블록을 시작 한다.
- LOCAL 변수는 IS 와 BEGIN 사이에 선언 한다.



docker exec -it oracle11g sqlplus



```
SQL> CREATE OR REPLACE PROCEDURE UPDATE_SALARY
  2  /* IN Argument */
  3  ( v_empno IN NUMBER )
  4
  5  IS
  6
  7  BEGIN
  8    UPDATE employees
  9    SET salary = salary * 1.1
 10   WHERE employee_id = v_empno;
 11   commit;
 12 END UPDATE_SALARY;
 13 /
```

Procedure created.

SQL>

02 프로시저 예제

```
docker exec -it oracle11g sqlplus

SQL> SELECT e.EMPLOYEE_ID , e.FIRST_NAME , e.SALARY
  2  FROM employees e
  3  WHERE e.EMPLOYEE_ID = 114;

EMPLOYEE_ID FIRST_NAME          SALARY
-----  -----
      114 Den                11000

SQL> EXECUTE UPDATE_SALARY(114);

PL/SQL procedure successfully completed.

SQL> SELECT e.EMPLOYEE_ID , e.FIRST_NAME , e.SALARY
  2  FROM employees e
  3  WHERE e.EMPLOYEE_ID = 114;

EMPLOYEE_ID FIRST_NAME          SALARY
-----  -----
      114 Den                12100

SQL>
```

02 프로시저 예제

DBeaver 21.0.0 - UPDATE_SALARY

Auto Oracle - hr HR UPDATE_SALARY

Properties Oracle - hr Schemas HR Procedures UPDATE_SALARY

Name: UPDATE_SALARY Valid

Arguments Declaration

```
CREATE OR REPLACE PROCEDURE HR.UPDATE_SALARY
/* IN Argument */
( v_empno IN NUMBER )

IS

BEGIN
    UPDATE employees
    SET salary = salary * 1.1
    WHERE employee_id = v_empno;
    commit;
END UPDATE_SALARY;
```

Oracle - hr - localhost:1521

Schemas

- ANONYMOUS
- APEX_040000
- APEX_PUBLIC_USER
- FLOW_FILES
- HR
 - Tables
 - Views
 - Materialized Views
 - Indexes
 - Sequences
 - Queues
 - Types
 - Packages
- Procedures
 - ADD_JOB_HISTORY
 - EMP_INFO
 - FC_UPDATE_SALARY
 - ROWTYPE_TEST
 - SECURE DML
 - UPDATE_SALARY
 - TABLE_TEST
- Synonyms

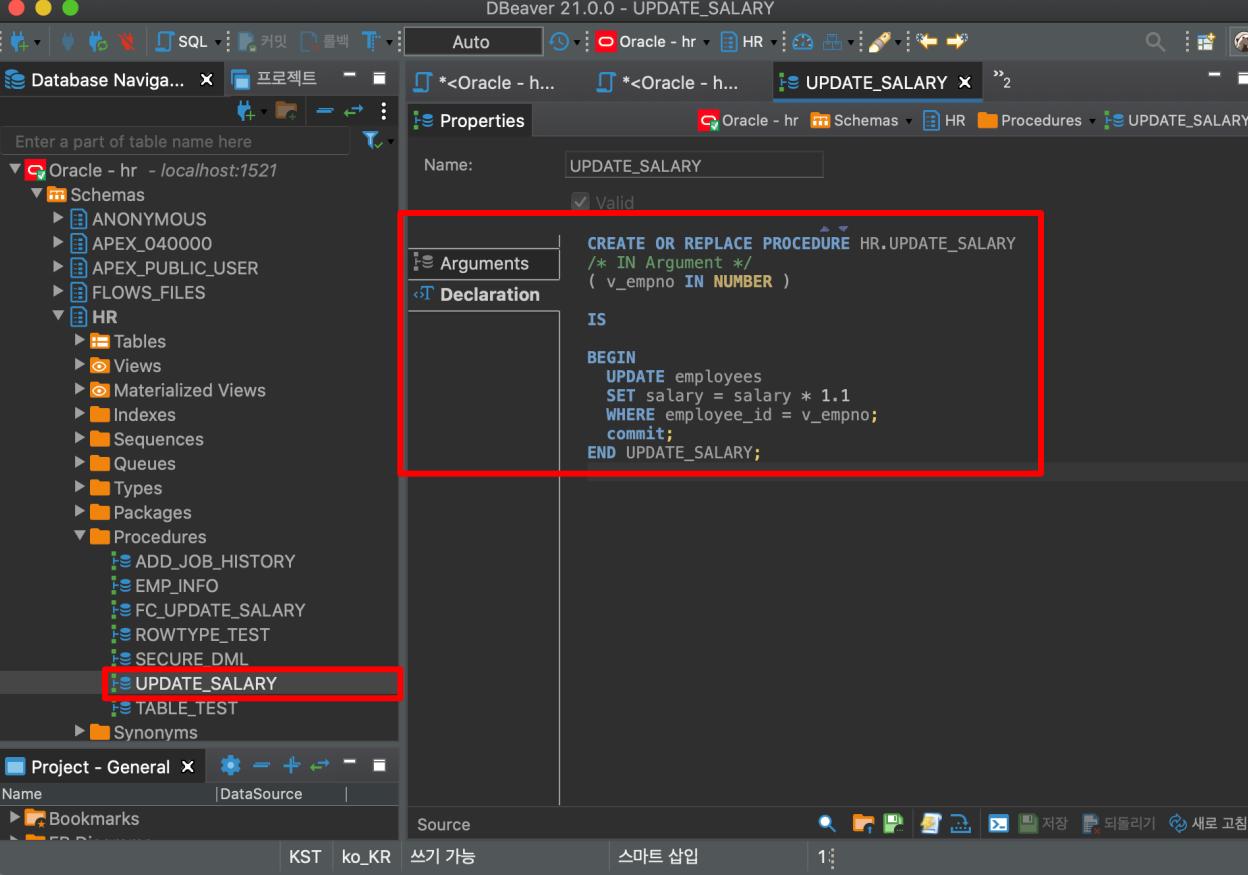
Project - General

Name: Data Source:

Source

Bookmarks

KST ko_KR 쓰기 가능 스마트 삽입 1%



02 함수(Function)란?

- 보통 값을 계산하고 결과 값을 반환하기 위해서 함수를 많이 사용 한다
- 대부분 구성이 프로시저와 유사 하지만 IN 파라미터만 사용 할 수 있다
- 반드시 반환 될 값의 데이터 타입을 RETURN문에 선언해야 한다
- 또한 PL/SQL블록 내에서 RETURN문을 통해서 반드시 값을 반환해야 한다

02 Function 문법

CREATE OR REPLACE FUNCTION function name

[(argument)]

RETURN datatype

-- datatype 은 반환되는 값의 datatype 임

IS

[변수의 선언]

BEGIN

[PL/SQL Block]

-- return 문이 꼭 존재해야 함

RETURN 변수;

END;

02 프로시저 예제

```
docker exec -it oracle11g sqlplus

SQL> CREATE OR REPLACE FUNCTION FC_UPDATE_SALARY
  2  (v_empno IN NUMBER)
  3  RETURN NUMBER
  4
  5  IS
  6
  7  v_salary employees.salary%type;
  8
  9  BEGIN
 10
 11  UPDATE EMPLOYEES
 12  SET SALARY = SALARY * 1.1
 13  WHERE EMPLOYEE_ID = v_empno;
 14
 15  COMMIT;
 16
 17  SELECT SALARY
 18  INTO v_salary
 19  FROM EMPLOYEES
 20  WHERE EMPLOYEE_ID = v_empno;
 21
 22  RETURN v_salary;
 23
 24  END;
 25  /

Function created.

SQL> █
```

02 프로시저 예제

```
docker exec -it oracle11g sqlplus

SQL> select first_name, salary
  2  from employees where employee_id = 114;

FIRST_NAME          SALARY
-----  -----
Den                  12100

SQL> 
```

1. Function 호출결과를 저장할 변수 선언
2. Function 호출
3. PRINT 문을 사용하여 출력

```
docker exec -it oracle11g sqlplus

SQL> VAR salary NUMBER;
SQL> EXECUTE :salary := FC_UPDATE_SALARY(114);

PL/SQL procedure successfully completed.

SQL> PRINT salary;

SALARY
-----
12100

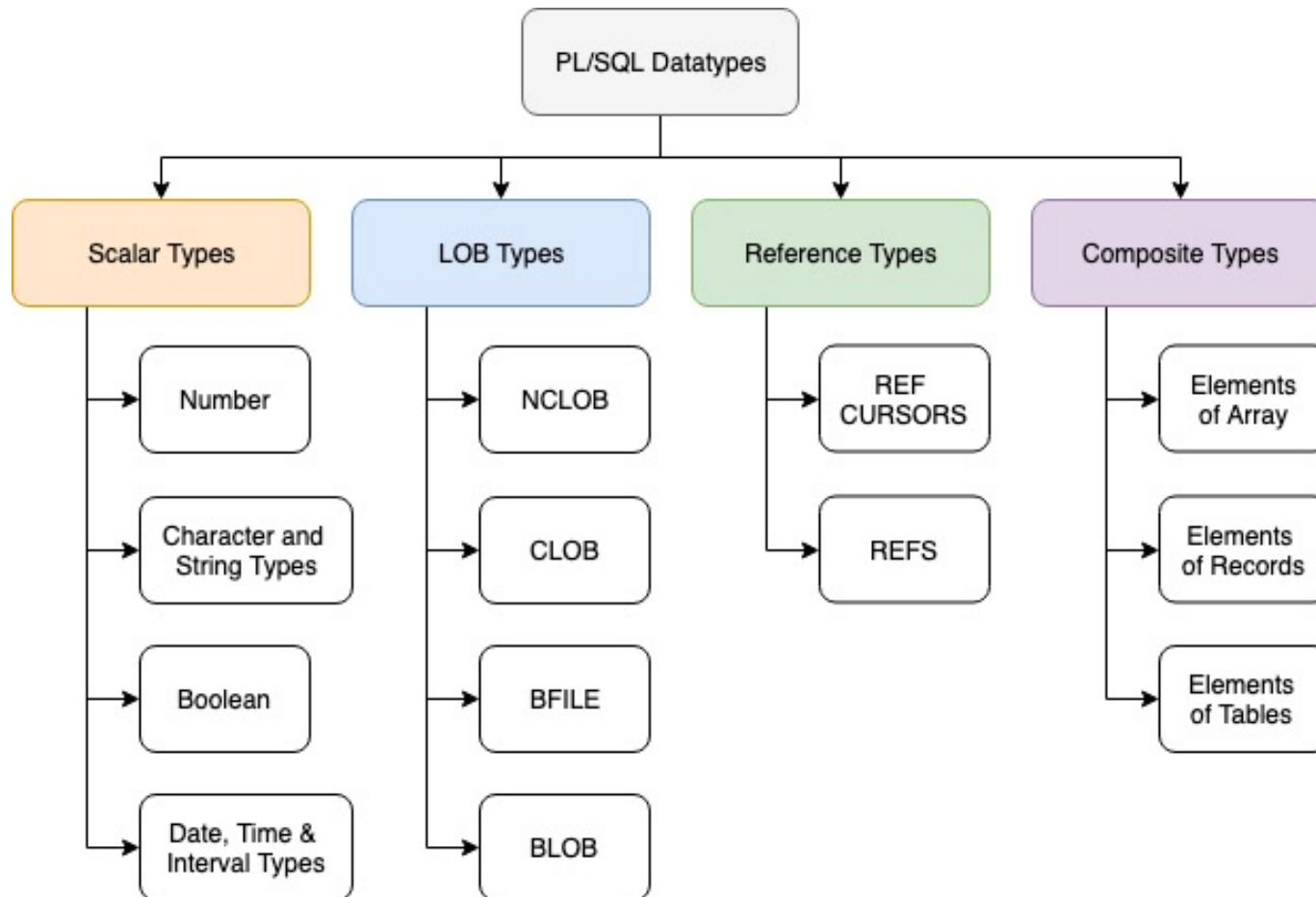
SQL> 
```

Oracle PL SQL



1. PL/SQL 개요
2. PROCEDURE VS FUNCTION
3. **PL/SQL Data Type**
4. PL/SQL 의 SQL
5. PL/SQL 제어문
6. SQL Cursor
7. Exception Handling
8. Package
9. Trigger

03 PL/SQL DataTypes



03 SCALAR DATA TYPE

스칼라 데이터타입에는 일반 단일 데이터타입의 변수와 %TYPE 데이터형 변수가 있다

일반변수 선언 문법

identifier [CONSTANT] 데이터타입 [NOT NULL] [:상수값 or 표현식];

- Identifier(변수)의 이름은 sql의 object명과 동일한 규칙을 따른다
- Identifier를 상수로 지정할때에는 CONSTANT라는 키워드를 명시하고 초기값을 할당
- NOT NULL이 정의되어 있으면 초기값을 반드시 지정하고, 정의되어 있지 않을 때는 생략 가능
- 초기값은 할당 연산자(:=)를 사용하여 정의
- 초기값을 정의하지 않으면 Identifier는 NULL값을 가지게 된다
- 일반적으로 한 줄에 한 개의 Identifier를 정의

03 SCALAR DATA TYPE

- 주요 SCALAR 변수의 데이터 타입

- **CHAR [(최대길이)]**

이 타입은 고정 길이의 문자를 저장하며 최대 32,767 바이트 값을 저장합니다. 기본 최소값은 1로 설정되어 있습니다.

- **VARCHAR2 (최대길이)**

이 타입은 가변 길이의 문자를 저장하며 최대 32,767 바이트 값을 저장합니다. 기본 값은 없습니다.

- **NUMBER [(전체 자리 수, 소수점 이하 자리 수)]**

이 타입은 전체 자리수와 소수점 이하의 자리 수를 가진 숫자입니다. 전체 자리수의 범위는 1부터 38까지, 소수점 이하 자리수의 범위는 -84 부터 127 까지입니다.

03 SCALAR DATA TYPE

- 주요 SCALAR 변수의 데이터 타입

- **BINARY_INTEGER**

이 타입은 $-2,147,483,647 \rightarrow 2,147,483,647$ 사이의 정수를 저장하는 타입입니다

- **PLS_INTEGER**

이 타입은 $-2,147,483,647 - 2,147,483,647$ 사이의 부호 있는 정수에 대한 기본 유형입니다. **PLS_INTEGER** 값은 **NUMBER** 값보다 저장 공간이 적게 필요하고 연산 속도가 더 빠릅니다. Oracle Database 11g에서는 **PLS_INTEGER** 및 **BINARY_INTEGER** 데이터 유형은 동일합니다
PLS_INTEGER 및 **BINARY_INTEGER** 값의 산술 연산은 **NUMBER** 값보다 빠릅니다

- **BOOLEAN**

이 타입은 논리적 계산에 사용 가능한 세 가지 값(**TRUE**, **FALSE**, **NULL**)중 하나를 저장하는 기본 유형입니다

- **BINARY_FLOAT**

이 타입은 IEEE 754 형식의 부동 소수점 수를 나타냅니다
값을 저장하기 위해 **5바이트가 필요합니다**

03 SCALAR DATA TYPE

- 주요 SCALAR 변수의 데이터 타입

- **BINARY_DOUBLE**

이 타입은 IEEE 754 형식의 부동 소수점 수를 나타냅니다
값을 저장하기 위해 9바이트가 필요합니다

- **DATE**

이 타입은 날짜 및 시간에 대한 기본 유형입니다. DATE 값은 자정 이후 경과한
시간을 초 단위로 포함합니다. 날짜의 범위는 4712 B.C. - 9999 A.D 사이입니다

- **TIMESTAMP**

이 타입은 DATE 데이터 유형을 확장하고 연도, 월, 일, 시, 분, 초 및 소수로 표시되는 초 단위를
저장합니다

구문은 TIMESTAMP[(precision)]이며 여기서 선택적 파라미터인 precision은 초 필드의 소수 부분
자릿수를 지정합니다

자릿수를 지정하려면 0 – 9 범위의 정수를 사용해야 합니다. 기본값은 6입니다

03 SCALAR DATA TYPE

- 주요 SCALAR 변수의 데이터 타입

- **TIMESTAMP WITH TIME ZONE**

이 타입은 TIMESTAMP 데이터 유형을 확장하고 시간대 변위를 포함합니다

시간대 변위는 로컬시간과 UTC(Coordinated Universal Time—이전의 그리니치 표준시)의 차이(시간과 분)입니다. 구문은 TIMESTAMP[(precision)] WITH TIME ZONE이며 여기서 선택적 파라미터 precision은 초 필드의 소수 부분 자릿수를 지정합니다

자릿수를 지정하려면 0~9 범위의 정수를 사용해야 하며, 기본값은 6입니다

- **TIMESTAMP WITH LOCAL TIME ZONE**

이 타입은 TIMESTAMP 데이터 유형을 확장하고 시간대 변위를 포함합니다

시간대 변위는 로컬시간과 UTC(Coordinated Universal Time—이전의 그리니치 표준시)의 차이(시간과 분)입니다. 구문은 TIMESTAMP[(precision)] WITH LOCAL TIME이며 여기서 선택적 파라미터 precision은 초 필드의 소수 부분 자릿수를 지정합니다

자릿수를 지정할 때 기호 상수 또는 변수는 사용할 수 없으며 0-9 범위의 정수 리터럴을 사용해야 하며, 기본값은 6입니다

이 데이터 유형은 데이터베이스 열에 값을 삽입하면 해당 값이 데이터베이스 시간대로 정규화되고 시간대 변위가 열에 저장되지 않는다는 점에서 TIMESTAMP WITH TIME ZONE 과 다릅니다 값을 검색할 때 Oracle 서버는 로컬 세션 시간대의 값을 반환합니다

03 SCALAR DATA TYPE

- 주요 SCALAR 변수의 데이터 타입

- **INTERVAL YEAR TO MONTH**

이 타입은 INTERVAL YEAR TO MONTH 데이터 유형은 연도와 월의 간격을 저장하거나 조작하는 데 사용됩니다 구문은 INTERVAL YEAR[(precision)] TO MONTH이며 여기서 precision은 연도 필드의 자릿수를 지정합니다 자릿수를 지정할 때 기호 상수 또는 변수는 사용할 수 없으며 0-4 범위의 정수 리터럴을 사용해야 하며, 기본값은 2입니다

- **INTERVAL DAY TO SECOND**

이 타입은 일, 시, 분, 초의 간격을 저장하거나 조작하는 데 사용됩니다
구문은 INTERVAL DAY[(precision1)] TO SECOND[(precision2)]이며 여기서 precision1 및 precision2는 각각 일 필드와 초 필드의 자릿수를 지정합니다
두 경우 모두 자릿수를 지정할 때 기호 상수 또는 변수는 사용할 수 없으며 0-9 범위의 정수 리터럴을 사용해야 하며, 기본값은 각각 2와 6입니다.

03 SCALAR DATA TYPE

일반변수 선언 예제

```
-- 숫자형 상수 선언(변할 수 없다)
v_price CONSTANT NUMBER(4,2) := 12.34 ;

v_name VARCHAR2(20) ;

v_Bir_Type CHAR(1) ;

-- NOT NULL의 TRUE로 초기화
v_flag BOOLEAN NOT NULL := TRUE ;

v_birthday DATE;
```

참조(Reference) 변수 선언

- v_no TableA.colA%TYPE -- tableA 테이블의 colA 와 동일한 데이터형으로 선언함
- v_nm TableA.colB%TYPE -- tableA 테이블의 colB 와 동일한 데이터형으로 선언함
- v_row TableB%ROWTYPE -- TableB 테이블의 여러 컬럼을 한꺼번에 저장할 변수로 선언함

03 Reference DATA TYPE

연습문제) 사원 테이블을 사용하여 사번을 입력받아 해당 사원의 사번, 이름(first_name), 월급을 출력하세요
ROWTYPE 변수를 사용할것 !

```
create or replace PROCEDURE PRINT_EMP
( v_input employees.EMPLOYEE_ID%TYPE )
IS
    v_row    employees%ROWTYPE;
BEGIN
    SELECT employee_id, first_name, salary, department_id
    INTO   v_row.employee_id, v_row.first_name, v_row.salary, v_row.department_id
    FROM EMPLOYEES
    WHERE EMPLOYEE_ID = v_input;

    dbms_output.put_line
    ( v_row.employee_id||' '||v_row.first_name||' '||v_row.salary||' '||v_row.department_id);

END PRINT_EMP;
/
```

SQL>execute PRINT_EMP(114);

03 SCALAR DATA TYPE

%TYPE 데이터형

- %TYPE 데이터형은 기술한 데이터베이스 테이블의 컬럼 데이터 타입을 모를 경우 사용할 수 있고, 코딩이후 데이터 베이스 컬럼의 데이터 타입이 변경될 경우 다시 수정할 필요가 없다
- 이미 선언된 다른 변수나 데이터베이스 컬럼의 데이터 타입을 이용하여 선언 한다
- 데이터베이스 테이블과 컬럼 그리고 이미 선언한 변수명이 %TYPE앞에 올수 있다

%TYPE 속성을 이용하여 얻을 수 있는 장점

- DB column definition을 정확히 알지 못하는 경우에 사용할 수 있다
- DB column definition이 변경 되어도 다시 PL/SQL을 수정할 필요가 없다

03 SCALAR DATA TYPE

```
docker exec -it oracle11g sqlplus
```

```

SQL> CREATE OR REPLACE PROCEDURE Emp_Info
  2  ( p_empno IN employees.employee_id%TYPE )
  3  IS
  4  v_empno employees.employee_id%TYPE;
  5  v_ename employees.first_name%TYPE;
  6  v_salary employees.salary%TYPE;
  7
  8 BEGIN
  9
 10 DBMS_OUTPUT.ENABLE;
 11
 12 SELECT employee_id, first_name, salary
 13 INTO v_empno, v_ename, v_salary
 14 FROM employees
 15 WHERE employee_id = p_empno;
 16
 17 DBMS_OUTPUT.PUT_LINE( 'EMP NO:'||v_empno );
 18 DBMS_OUTPUT.PUT_LINE( 'EMP NAME:'||v_ename );
 19 DBMS_OUTPUT.PUT_LINE( 'EMP SALARY:'||v_salary );
 20
 21 END;
 22 /

```

Procedure created.

```
SQL> █
```

%TYPE 사용 예제

```
docker exec -it oracle11g sqlplus
```

```

SQL> EXECUTE Emp_Info(114);
EMP NO:114
EMP NAME:Den
EMP SALARY:12100

PL/SQL procedure successfully completed.

SQL> █

```

03 Composite DataTypes

%ROWTYPE

하나 이상의 데이터값을 갖는 데이터 타입으로 배열과 비슷한 역할을 하고 재사용이 가능하다

%ROWTYPE 데이터형, PL/SQL테이블, 레코드는 복합 데이터 타입에 속한다

- 테이블이나 뷰 내부의 컬럼 데이터형, 크기, 속성 등을 그대로 사용 할 수 있다
- %ROWTYPE 앞에 오는 것은 데이터베이스 테이블 이름이다
- 지정된 테이블의 구조와 동일한 구조를 갖는 변수를 선언 할 수 있다
- 데이터베이스 컬럼들의 수나 DATATYPE을 알지 못할 때 편리 하다
- 테이블의 데이터 컬럼의 DATATYPE이 변경 될 경우 프로그램을 재수정할 필요가 없다

03 Composite DataTypes

%ROWTYPE 예제

```

SQL> CREATE OR REPLACE PROCEDURE RowType_Test
  ( p_empno IN employees.employee_id%TYPE )
IS
  v_emp employees%ROWTYPE;
BEGIN
  DBMS_OUTPUT.ENABLE;

  SELECT employee_id, first_name, hire_date
  INTO v_emp.employee_id, v_emp.first_name, v_emp.hire_date
  FROM employees
  WHERE employee_id = p_empno;

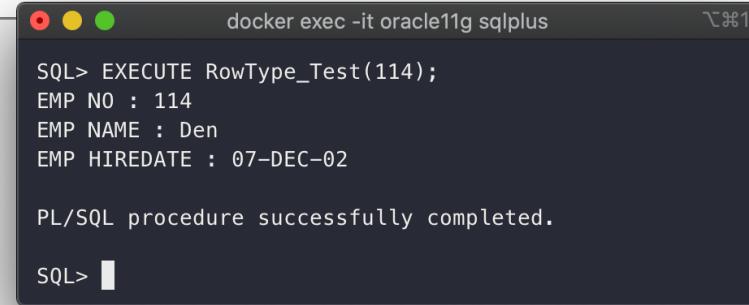
  DBMS_OUTPUT.PUT_LINE('EMP NO : '||v_emp.employee_id);
  DBMS_OUTPUT.PUT_LINE('EMP NAME : '||v_emp.first_name);
  DBMS_OUTPUT.PUT_LINE('EMP HIREDATE : '||v_emp.hire_date);

END;
/

```

Procedure created.

SQL>



```

docker exec -it oracle11g sqlplus
SQL> EXECUTE RowType_Test(114);
EMP NO : 114
EMP NAME : Den
EMP HIREDATE : 07-DEC-02

PL/SQL procedure successfully completed.

SQL>

```

03 Composite DataTypes

PL/SQL 테이블

- PL/SQL에서의 테이블은 오라클 SQL에서의 테이블과는 다르다 (PL/SQL에서의 테이블은 일종의 일차원 배열이다)
- 테이블은 크기에 제한이 없으며 그 ROW의 수는 데이터가 들어옴에 따라 자동 증가 한다
BINARY_INTEGER 타입의 인덱스 번호로 순서가 정해진다
- 하나의 테이블에 한 개의 컬럼 데이터를 저장 한다

PL/SQL 테이블 문법

```
TYPE table_name IS TABLE OF datatype INDEX BY BINARY_INTEGER;  
identifier table_name;
```

-- 사용예제

```
TYPE prdname_table IS TABLE OF VARCHAR2(30) INDEX BY BINARY_INTEGER;
```

-- prdname_table 테이블타입으로 prdname_tab 변수를 선언해서 사용
prdname_tab prdname_table

03 Composite DataTypes : PL/SQL 테이블

```
CREATE OR REPLACE PROCEDURE TABLE_TEST
(v_deptno IN employees.DEPARTMENT_ID%TYPE)
IS
    -- 테이블의 선언
    TYPE empno_table IS TABLE OF employees.employee_id%TYPE INDEX BY BINARY_INTEGER;
    TYPE ename_table IS TABLE OF employees.first_name%TYPE INDEX BY BINARY_INTEGER;
    TYPE sal_table     IS TABLE OF employees.salary%TYPE INDEX BY BINARY_INTEGER;

    -- 테이블타입으로 변수 선언
    empno_tab empno_table ;
    ename_tab ename_table ;
    sal_tab   sal_table;
    i BINARY_INTEGER := 0;
BEGIN
    DBMS_OUTPUT.ENABLE;

    -- FOR 루프 사용
    -- 여기서 emp_list는 ( BINARY_INTEGER형 변수로) 1씩 증가
    FOR emp_list IN ( SELECT employee_id, first_name, salary
                        FROM employees
                       WHERE department_id = v_deptno ) LOOP
        i := i + 1;
```

03 Composite DataTypes : PL/SQL 테이블

```
-- 테이블 변수에 검색된 결과를 넣는다  
empno_tab(i) := emp_list.employee_id ;  
ename_tab(i) := emp_list.first_name ;  
sal_tab(i) := emp_list.salary ;  
END LOOP;  
  
-- 1부터 i까지 FOR 문을 실행  
FOR cnt IN 1..i LOOP  
    -- TABLE변수에 넣은 값을 뿌려줌  
    DBMS_OUTPUT.PUT_LINE( '사원번호 :' || empno_tab(cnt) );  
    DBMS_OUTPUT.PUT_LINE( '사원이름 :' || ename_tab(cnt) );  
    DBMS_OUTPUT.PUT_LINE( '사원급여 :' || sal_tab(cnt));  
END LOOP;  
END TABLE_TEST;
```

03 Composite DataTypes : PL/SQL 테이블

```

CREATE OR REPLACE PROCEDURE TABLE_TEST
(v_deptno IN employees.department_id %TYPE)
IS
    -- 테이블의 선언
    TYPE empno_table IS TABLE OF employees.employee_id%TYPE INDEX BY BINARY_INTEGER;
    TYPE ename_table IS TABLE OF employees.first_name%TYPE INDEX BY BINARY_INTEGER;
    TYPE sal_table   IS TABLE OF employees.salary%TYPE INDEX BY BINARY_INTEGER;

    -- 테이블타입으로 변수 선언
    empno_tab empno_table ;
    ename_tab ename_table ;
    sal_tab   sal_table;
    i BINARY_INTEGER := 0;
BEGIN
    DBMS_OUTPUT.ENABLE;

    -- FOR 루프 사용
    -- 여기서 emp_list는 ( BINARY_INTEGER형 변수로 ) 1씩 증가
    FOR emp_list IN ( SELECT employee_id, first_name, salary
                        FROM employees
                       WHERE department_id = v_deptno ) LOOP

        i := i + 1;
        -- 테이블 변수에 검색된 결과를 넣는다
        empno_tab(i) := emp_list.employee_id ;
        ename_tab(i) := emp_list.first_name ;
        sal_tab(i) := emp_list.salary ;
    END LOOP;

    -- 1부터 i까지 FOR 문을 실행
    FOR cnt IN 1..i LOOP
        -- TABLE변수에 넣은 값을 뿌려줌
        DBMS_OUTPUT.PUT_LINE( '사원번호 : ' || empno_tab(cnt) );
        DBMS_OUTPUT.PUT_LINE( '사원이름 : ' || ename_tab(cnt) );
        DBMS_OUTPUT.PUT_LINE( '사원급여 : ' || sal_tab(cnt));
    END LOOP;
END TABLE_TEST;

```

```

docker exec -it oracle11g sqlplus

SQL> set serveroutput on;
SQL> execute table_test(100);
EMP NO : 108
EMP NAME : Nancy
EMP SALARY : 12008
EMP NO : 109
EMP NAME : Daniel
EMP SALARY : 9000
EMP NO : 110
EMP NAME : John
EMP SALARY : 8200
EMP NO : 111
EMP NAME : Ismael
EMP SALARY : 7700
EMP NO : 112
EMP NAME : Jose Manuel
EMP SALARY : 7800
EMP NO : 113
EMP NAME : Luis
EMP SALARY : 6900

PL/SQL procedure successfully completed.

SQL>

```

03 Composite DataTypes : PL/SQL RECORD

PL/SQL RECORD

- PL/SQL레코드는 여러개의 데이터 타입을 갖는 **변수들의 집합**이다
- 스칼라, RECORD, 또는 PL/SQL TABLE datatype중 하나 이상의 요소로 구성 된다
- 논리적 단위로서 필드 집합을 처리할 수 있도록 해 준다
- PL/SQL 테이블과 다르게 개별 필드의 이름을 부여할 수 있고, 선언시 초기화가 가능하다.

03 Composite DataTypes : PL/SQL RECORD

PL/SQL RECORD 문법

```
TYPE record_name IS RECORD  
( fieldname datatype [NOT NULL {: I DEFAULT} expr ],  
  fieldname datatype [NOT NULL {: I DEFAULT} expr ],  
  fieldname datatype [NOT NULL {: I DEFAULT} expr ] );
```

```
Identifier record_name ;
```

PL/SQL RECORD 선언 예제

```
TYPE record_test IS RECORD  
( record_empno NUMBER,  
  record_ename VARCHAR2(30),  
  record_sal    NUMBER );
```

```
prd_record record_test ;
```

03 Composite DataTypes : PL/SQL RECORD

```

CREATE OR REPLACE PROCEDURE Record_Test
( p_empno IN employees.employee_id%TYPE )
IS
    -- 하나의 레코드의 세가지의 변수타입 선언
    TYPE emp_record IS RECORD
        (v_empno      NUMBER,
         v_ename      VARCHAR2(30),
         v_hiredate   DATE);

    emp_rec emp_record ;
BEGIN
    DBMS_OUTPUT.ENABLE;

    -- 레코드의 사용
    SELECT employee_id, first_name, hire_date
    INTO emp_rec.v_empno, emp_rec.v_ename, emp_rec.v_hiredate
    FROM employees
    WHERE employee_id = p_empno;

    DBMS_OUTPUT.PUT_LINE( 'EMP NO : ' || emp_rec.v_empno );
    DBMS_OUTPUT.PUT_LINE( 'EMP NAME : ' || emp_rec.v_ename );
    DBMS_OUTPUT.PUT_LINE( 'EMP HIREDATE: ' || emp_rec.v_hiredate);
END;

```

```

docker exec -it oracle11g sqlplus
SQL> execute record_test(100);
EMP NO : 100
EMP NAME : Steven
EMP HIREDATE: 17-JUN-03

PL/SQL procedure successfully completed.

SQL>

```

03 Composite DataTypes : Table of Record

- PL/SQL TABLE 변수 선언과 비슷하며 데이터타입을 %ROWTYPE으로 선언하면 된다
- PL/SQL TABLE과 RECORD의 복합 기능을 한다

PL/SQL Table of Record 문법

```
TYPE Table_of_Record_Name IS TABLE OF datatype INDEX BY BINARY_INTEGER;  
  
Identifier record_name ;
```

PL/SQL Table of Record 선언 예제

```
TYPE dept_table_type IS TABLE OF department%ROWTYPE INDEX BY BINARY_INTEGER;  
  
dept_table dept_table_type;
```

03 Composite DataTypes : Table of Record

```

CREATE OR REPLACE PROCEDURE TABLE_OF_RECORD
IS
    i BINARY_INTEGER := 0;
    -- PL/SQL Table of Record의 선언
    TYPE dept_table_type IS TABLE OF departments%ROWTYPE INDEX BY BINARY_INTEGER;
    dept_table dept_table_type;
BEGIN
    FOR dept_list IN (SELECT * FROM departments) LOOP
        i:= i+1;

        -- TABLE OF RECORD에 데이터 보관
        dept_table(i).department_id := dept_list.department_id ;
        dept_table(i).department_name := dept_list.department_name ;
        dept_table(i).location_id := dept_list.location_id ;
    END LOOP;

    FOR cnt IN 1..i LOOP
        -- 데이터 출력
        DBMS_OUTPUT.PUT_LINE( ' DEPT NO : ' || dept_table(cnt).department_id ||
                             ' DEPT NAME : ' || dept_table(cnt).department_name ||
                             ' DEPT LOC : ' || dept_table(cnt).location_id );
    END LOOP;
END;

```

docker exec -it oracle11g sqlplus /nolog
SQL*Plus: Release 11.2.0.1.0 Production on Tue Dec 10 10:45:20 2019
Copyright (c) 1982, 2009, Oracle. All rights reserved.
Connected to:
Oracle Database 11g Enterprise Edition Release 11.2.0.1.0 - 64bit Production
With the Partitioning, OLAP, Data Mining and Real Application Testing options
SQL> execute table_of_record;
DEPT NO : 10 DEPT NAME : Administration DEPT LOC :
DEPT NO : 20 DEPT NAME : Marketing DEPT LOC :
DEPT NO : 30 DEPT NAME : Purchasing DEPT LOC :
DEPT NO : 40 DEPT NAME : Human Resources DEPT LOC :
DEPT NO : 50 DEPT NAME : Shipping DEPT LOC :
DEPT NO : 60 DEPT NAME : IT DEPT LOC : 1400
DEPT NO : 70 DEPT NAME : Public Relations DEPT LOC :
DEPT NO : 80 DEPT NAME : Sales DEPT LOC : 1300
DEPT NO : 90 DEPT NAME : Executive DEPT LOC :
DEPT NO : 100 DEPT NAME : Finance DEPT LOC :
DEPT NO : 110 DEPT NAME : Accounting DEPT LOC :
DEPT NO : 120 DEPT NAME : Treasury DEPT LOC :
DEPT NO : 130 DEPT NAME : Corporate Tax DEPT LOC :
DEPT NO : 140 DEPT NAME : Control And Credit DEPT LOC :
DEPT NO : 150 DEPT NAME : Shareholder Services DEPT LOC :
DEPT NO : 160 DEPT NAME : Benefits DEPT LOC :
DEPT NO : 170 DEPT NAME : Manufacturing DEPT LOC :
DEPT NO : 180 DEPT NAME : Construction DEPT LOC :
DEPT NO : 190 DEPT NAME : Contracting DEPT LOC :
DEPT NO : 200 DEPT NAME : Operations DEPT LOC :
DEPT NO : 210 DEPT NAME : IT Support DEPT LOC :
DEPT NO : 220 DEPT NAME : NOC DEPT LOC : 1700
DEPT NO : 230 DEPT NAME : IT Helpdesk DEPT LOC :
DEPT NO : 240 DEPT NAME : Government Sales DEPT LOC :
DEPT NO : 250 DEPT NAME : Retail Sales DEPT LOC :
DEPT NO : 260 DEPT NAME : Recruiting DEPT LOC :
DEPT NO : 270 DEPT NAME : Payroll DEPT LOC :
PL/SQL procedure successfully completed.

Oracle PL SQL



1. PL/SQL 개요
2. PROCEDURE VS FUNCTION
3. PL/SQL Data Type
4. **PL/SQL 의 SQL**
5. PL/SQL 제어문
6. SQL Cursor
7. Exception Handling
8. Package
9. Trigger

04 PL/SQL SQL : INSERT

- PL/SQL에서의 INSERT 문은 SQL과 비슷하다

```

CREATE OR REPLACE PROCEDURE Insert_Test
( v_empno IN employees.employee_id%TYPE,
  v_ename IN employees.last_name%TYPE,
  v_email IN employees.email%TYPE,
  v_job    IN employees.job_id%TYPE,
  v_deptno IN employees.department_id%TYPE )
IS
BEGIN
  DBMS_OUTPUT.ENABLE;          CALL HR.INSERT_TEST(500, 'BatMan', 'batman@disney.com', 'MK_MAN', 20);
  INSERT INTO employees
  (employee_id, last_name, email, job_id, hire_date, department_id)
  VALUES(v_empno, v_ename, v_email, v_job, sysdate, v_deptno);
  COMMIT;
  DBMS_OUTPUT.PUT_LINE( 'EMP NO : ' || v_empno );
  DBMS_OUTPUT.PUT_LINE( 'EMP NAME : ' || v_ename );
  DBMS_OUTPUT.PUT_LINE( 'EMP EMAIL : ' || v_email );
  DBMS_OUTPUT.PUT_LINE( 'EMP JOB : ' || v_job );
  DBMS_OUTPUT.PUT_LINE( 'EMP DEPT : ' || v_deptno );
  DBMS_OUTPUT.PUT_LINE( 'EMP DATA INSERT SUCCESS!' );
END ;

```

The screenshot shows the execution of the PL/SQL procedure. The command `CALL HR.INSERT_TEST(500, 'BatMan', 'batman@disney.com', 'MK_MAN', 20);` is highlighted with a red box. The output window displays the inserted employee data and a success message.

Output
EMP NO : 500 EMP NAME : BatMan EMP EMAIL : batman@disney.com EMP JOB : MK_MAN EMP DEPT : 20 EMP DATA INSERT SUCCESS!

04 PL/SQL SQL : UPDATE

```

CREATE OR REPLACE PROCEDURE UPDATE_TEST
( v_empno IN employees.employee_id%TYPE,      -- 급여를 수정한 사원의 사번
  v_sal    IN employees.salary%TYPE )           -- 수정할 급여
IS
  -- 수정 데이터를 확인하기 위한 변수 선언
  v_emp employees%ROWTYPE;
BEGIN DBMS_OUTPUT.ENABLE;
  -- 급여 수정
  UPDATE employees
  SET salary = v_sal
  WHERE employee_id = v_empno ;
  COMMIT;

  DBMS_OUTPUT.PUT_LINE('Data Update Success ');

  -- 수정된 데이터 확인하기 위해 검색
  SELECT employee_id, last_name, salary
  INTO v_emp.employee_id, v_emp.last_name, v_emp.salary
  FROM employees
  WHERE employee_id = v_empno ;
  DBMS_OUTPUT.PUT_LINE( ' **** Confirm Update Data **** ' );
  DBMS_OUTPUT.PUT_LINE( 'EMP NO : ' || v_emp.employee_id );
  DBMS_OUTPUT.PUT_LINE( 'EMP NAME : ' || v_emp.last_name );
  DBMS_OUTPUT.PUT_LINE( 'EMP SALARY : ' || v_emp.salary );
END ;

```

```

SQL> execute update_test(500, 5000);
Data Update Success
**** Confirm Update Data ****
EMP NO : 500
EMP NAME : BatMan
EMP SALARY : 5000

PL/SQL procedure successfully completed.

SQL> 

```

04 PL/SQL SQL : DELETE

```

CREATE OR REPLACE PROCEDURE DELETE_TEST
( p_empno IN employees.employee_id%TYPE )
IS
    -- 삭제 데이터를 확인하기 레코드 선언
    TYPE del_record IS RECORD
    ( v_empno      employees.employee_id%TYPE,
      v_ename       employees.last_name%TYPE,
      v_hiredate   employees.hire_date%TYPE ) ;

    v_emp del_record ;
BEGIN
    DBMS_OUTPUT.ENABLE;
    -- 삭제된 데이터 확인용 쿼리
    SELECT employee_id, last_name, hire_date
    INTO v_emp.v_empno, v_emp.v_ename, v_emp.v_hiredate
    FROM employees
    WHERE employee_id = p_empno ;

    DBMS_OUTPUT.PUT_LINE( 'EMP NO : ' || v_emp.v_empno );
    DBMS_OUTPUT.PUT_LINE( 'EMP NAME : ' || v_emp.v_ename );
    DBMS_OUTPUT.PUT_LINE( 'EMP HDATE : ' || v_emp.v_hiredate );

    -- 삭제 쿼리
    DELETE
    FROM employees
    WHERE employee_id = p_empno ;

    COMMIT;

    DBMS_OUTPUT.PUT_LINE( 'DATA DELETE SUCCESS ' );
END;

```

docker exec -it oracle11g sqlplus

```

SQL> execute delete_test(500);
EMP NO : 500
EMP NAME : BatMan
EMP HDATE : 14-MAR-21
DATA DELETE SUCCESS

PL/SQL procedure successfully completed.

SQL>

```