# Attribute

# %TYPE Attribute

The %TYPE attribute lets you declare a constant, variable, field, or parameter to be of the same data type a previously declared variable, field, record, nested table, or database column. If the referenced item changes, your declaration is automatically updated.

Example:

lname employees.last\_name%TYPE;

fname employees.first\_name%TYPE;

# %ROWTYPE Attribute

CREATE TABLE employees\_temp (

empid NUMBER(6) PRIMARY KEY,

deptid NUMBER(6),

deptname VARCHAR2(30)

);

-- *Declaring and Assigning Values to Variables*

emprec employees\_temp%ROWTYPE;

……..

emprec.empid := NULL;

emprec.deptid := 50;

# Record Type

--defining a type

TYPE Timerec IS RECORD (

hours SMALLINT,

minutes SMALLINT

);

--declaring a variable

time Timerec;

# PL/SQL Block

DECLARE -- Declarative part (optional)

Declarations of local types, variables, & subprograms

BEGIN --Executable part (required)

Statements (which can use items declared in declarative part)

EXCEPTION -- Exception-handling part (optional)

Exception handlers for exceptions raised in executable part

END;

# *Declaring Variables in PL/SQL*

DECLARE

part\_number NUMBER(6); -- SQL data type

part\_name VARCHAR2(20); -- SQL data type

in\_stock BOOLEAN; -- PL/SQL-only data type

part\_price NUMBER(6,2); -- SQL data type

part\_description VARCHAR2(50); -- SQL data type

BEGIN

NULL;

END;

***Assigning Values to Variables With the Assignment Operator***

DECLARE

wages NUMBER;

country VARCHAR2(128);

emp\_rec1 employees%ROWTYPE;

emp\_rec2 employees%ROWTYPE;

BEGIN

wages := (hours\_worked \* hourly\_salary) + bonus;

country := UPPER('Canada');

emp\_rec1.first\_name := 'Antonio';

emp\_rec1.last\_name := 'Ortiz';

emp\_rec2 := emp\_rec1;

END;

***Assigning Values to Variables by SELECTing INTO***

DECLARE

bonus NUMBER(8,2);

emp\_id NUMBER(6):= 100;

BEGIN

SELECT salary \* 0.10 INTO bonus

FROM employees

WHERE employee\_id = emp\_id;

END;

# IF-THEN-ELSIF Statement

DECLARE

sales NUMBER(8,2) := 20000;

bonus NUMBER(6,2);

emp\_id NUMBER(6) := 120;

BEGIN

IF sales > 50000 THEN

bonus := 1500;

ELSIF sales > 35000 THEN

bonus := 500;

ELSE

bonus := 100;

END IF;

UPDATE employees SET salary = salary + bonus

WHERE employee\_id = emp\_id;

END;

# LOOP Statements

**Basic LOOP**

DECLARE

x NUMBER := 0;

BEGIN

LOOP

DBMS\_OUTPUT.PUT\_LINE ('Inside loop: x = ' || TO\_CHAR(x));

x := x + 1;

IF x > 3 THEN

EXIT;

END IF;

END LOOP;

-- After EXIT, control resumes here

DBMS\_OUTPUT.PUT\_LINE (' After loop: x = ' || TO\_CHAR(x));

END;

**FOR LOOP Statement**

BEGIN

FOR i IN 1..3 LOOP

DBMS\_OUTPUT.PUT\_LINE (TO\_CHAR(i));

END LOOP;

END;

# PL/SQL Error Handling

EXCEPTION

WHEN *ex\_name\_1* THEN *statements\_1* -- Exception handler

WHEN *ex\_name\_2* OR *ex\_name\_3* THEN *statements\_2* -- Exception handler

WHEN OTHERS THEN *statements\_3* -- Exception handler

END;

## **Predefined Exceptions**

| **Exception Name** | **Error Code** |
| --- | --- |
| ACCESS\_INTO\_NULL | -6530 |
| CASE\_NOT\_FOUND | -6592 |
| COLLECTION\_IS\_NULL | -6531 |
| CURSOR\_ALREADY\_OPEN | -6511 |
| DUP\_VAL\_ON\_INDEX | -1 |
| INVALID\_CURSOR | -1001 |
| INVALID\_NUMBER | -1722 |
| LOGIN\_DENIED | -1017 |
| NO\_DATA\_FOUND | +100 |
| NO\_DATA\_NEEDED | -6548 |
| NOT\_LOGGED\_ON | -1012 |
| PROGRAM\_ERROR | -6501 |
| ROWTYPE\_MISMATCH | -6504 |
| SELF\_IS\_NULL | -30625 |
| STORAGE\_ERROR | -6500 |
| SUBSCRIPT\_BEYOND\_COUNT | -6533 |
| SUBSCRIPT\_OUTSIDE\_LIMIT | -6532 |
| SYS\_INVALID\_ROWID | -1410 |
| TIMEOUT\_ON\_RESOURCE | -51 |
| TOO\_MANY\_ROWS | -1422 |
| VALUE\_ERROR | -6502 |
| ZERO\_DIVIDE | -1476 |

Example:

NO\_DATA\_FOUND exception

DECLARE

mgr\_id employees.manager\_id%TYPE;

starting\_empid employees.employee\_id%TYPE := 120;

BEGIN

SELECT manager\_id INTO mgr\_id FROM employees

WHERE employee\_id = starting\_empid;

--do something here

EXCEPTION

WHEN NO\_DATA\_FOUND THEN

--do something here

DBMS\_OUTPUT.PUT\_LINE ('not found');

COMMIT;

END;

**ZERO\_DIVIDE exception**

DECLARE

stock\_price NUMBER := 9.73;

**net\_earnings NUMBER := 0**;

pe\_ratio NUMBER;

BEGIN

pe\_ratio := **stock\_price / net\_earnings**; -- raises ZERO\_DIVIDE exception

DBMS\_OUTPUT.PUT\_LINE('Price/earnings ratio = ' || pe\_ratio);

**EXCEPTION**

**WHEN ZERO\_DIVIDE** THEN

DBMS\_OUTPUT.PUT\_LINE('Company had zero earnings.');

pe\_ratio := NULL;

END;

* ***Anonymous Block Avoids ZERO\_DIVIDE***

DECLARE

stock\_price NUMBER := 9.73;

net\_earnings NUMBER := 0;

pe\_ratio NUMBER;

BEGIN

IF(net\_earnings = 0) THEN

pe\_ratio:=NULL;

ELSE net\_earnings

pe\_ratio:= stock\_price / net\_earnings;

END IF;

END;

## Defining Your Own PL/SQL Exceptions

DECLARE ---------- sub-block begins

**past\_due** **EXCEPTION**; -- this declaration prevails

due\_date DATE := SYSDATE - 1;

todays\_date DATE := SYSDATE;

BEGIN

IF due\_date < todays\_date THEN

RAISE past\_due; -- this is not handled

END IF;

EXCEPTION

WHEN past\_due THEN

DBMS\_OUTPUT.PUT\_LINE('Handling PAST\_DUE exception.');

WHEN OTHERS THEN

DBMS\_OUTPUT.PUT\_LINE

('Could not recognize PAST\_DUE\_EXCEPTION in this scope.');

END;

# PL/SQL Subprograms

A PL/SQL **subprogram** is a named PL/SQL block that can be invoked repeatedly. If the subprogram has parameters, their values can differ for each invocation.

A subprogram is either a procedure or a function. Typically, you use a procedure to perform an action and a function to compute and return a value.

CREATE OR REPLACE PROCEDURE double (original IN VARCHAR2,

new\_string OUT VARCHAR2)

IS

-- Declarative part of procedure (optional) goes here

BEGIN

-- Executable part of procedure begins

new\_string := original || ' + ' || original;

-- Executable part of procedure ends

-- Exception-handling part of procedure (optional) begins

EXCEPTION

WHEN VALUE\_ERROR THEN

DBMS\_OUTPUT.PUT\_LINE('Output buffer not long enough.');

END;

A function has the same structure as a procedure, except that:

* A function heading must include a RETURN **clause**, which specifies the data type of the value that the function returns. (A procedure heading cannot have a RETURN clause.)
* In the executable part of a function, every execution path must lead to a RETURN **statement**. Otherwise, the PL/SQL compiler issues a compile-time warning. (In a procedure, the RETURN statement is optional and not recommended.)

PROCEDURE raise\_salary ( emp\_id NUMBER, amount NUMBER)

IS

BEGIN

IF emp\_id IS NULL THEN

RETURN;

END IF;

UPDATE employees SET salary = salary + amount WHERE employee\_id = emp\_id;

END raise\_salary;

FUNCTION compute\_bonus (emp\_id NUMBER, bonus NUMBER) RETURN NUMBER

IS

emp\_sal NUMBER;

BEGIN

SELECT salary INTO emp\_sal FROM employees WHERE employee\_id = emp\_id;

RETURN emp\_sal + bonus;

END compute\_bonus;