1. 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;
```

2. 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;
```

3. 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;
```

4. 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;
```

5. 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

Exception Name	Error Code
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;
```

6. 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;
```