Data type synonyms

Data types have synonyms for compatibility with **non-Oracle** data sources such as **IBM Db2**, **SQL Server**.

It is not a good practice to use data type synonym **unless** you are accessing a **non-Oracle Database**.

شركة تحالف الإمارات للحلول التقنية ذ.م.م.
TAHALUF AL EMARAT TECHNICAL SOLUTIONS L.L.C.

Data type synonyms

Data Type	Synonyms
NUMBER	DEC, DECIMAL, DOUBLE PRECISION, FLOAT, INTEGER, INT, NUMERIC, REAL, SMALLINT
CHAR	CHARACTER, STRING
VARCHAR2	VARCHAR

Variables

In PL/SQL, a **variable** is named storage location that stores a value of a particular data type. The value of the variable changes through the program. Before using a variable, you must declare it in the declaration section of a block.

Variables

The syntax for a variable declaration is as follows:

```
variable_name datatype [NOT NULL] [:= initial_value];
```

Variables

In the last syntax:

- **First**, specify the name of the variable. The name of the variable should be as descriptive as possible, e.g., **l_total_sales**, **l_credit_limit**, and **l_sales_revenue**.
- **Second**, choose an appropriate data type for the variable, depending on the kind of value which you want to store, for example, **number**, **character**, **Boolean**, and **datetime**.
- **By convention**, **local** variable names should start with **l_** and **global** variable names should have a prefix of **g_** .

Variables

```
DECLARE
   l_total_sales NUMBER(15,2);
   l_credit_limit NUMBER (10,0);
   l_contact_name VARCHAR2(255);
BEGIN
   NULL;
END;
```

Variables

Default values:

PL/SQL allows you to set a default value for a variable at the declaration time. To assign a default value to a variable, you use the assignment operator (:=) or the DEFAULT keyword.

Variables

شركة تحالف الإمارات للحلول التقنية ذ.م.م.
TAHALUF AL EMARAT TECHNICAL SOLUTIONS L.L.C.

```
DECLARE
 l_product_name VARCHAR2( 100 ) := 'Laptop';
BEGIN
 NULL;
END;
```

It is equivalent to the following block:

```
DECLARE
  l_product_name VARCHAR2(100) DEFAULT 'Laptop';
BEGIN
  NULL;
END;
```

Variables

NOT NULL constraint:

If you impose the **NOT NULL** constraint on a value, then the variable cannot accept a NULL value.

Besides, a variable declared with the NOT NULL must be initialized with a non-null value.

Note that PL/SQL treats a zero-length string as a NULL value.

Variables

```
DECLARE
  l_shipping_status VARCHAR2( 25 ) NOT NULL := 'Shipped';
BEGIN
  l_shipping_status := '';
END;
```

It is equivalent to the following block:

```
ORA-06502: PL/SQL: numeric or value error
```

Variables

شركة تحالف الإمارات للحلول التقنية ذ.م.م.
TAHALUF AL EMARAT TECHNICAL SOLUTIONS L.L.C.

Anchored declarations:

Typically, you declare a variable and select a value from a table column to this variable.

If the data type of the table column changes, you must adjust the program to make it work with the new type.

PL/SQL allows you to declare a variable whose data type anchor to a table column or another

Variables

```
DECLARE
 l_name DEPT.DNAME%TYPE;
 l_loc DEPT.loc%TYPE;
BEGIN
 SELECT
  DNAMe, loc
 INTO
  l_name, l_loc
 FROM
  DEPT
 WHERE
  DEPTNO = 3;
 DBMS_OUTPUT.PUT_LINE(l_name || ':' || l_loc );
END;
/
```

Variables

```
DECLARE
   l_Name student.name%TYPE;
   l_average_mark student.mark%TYPE;
   l_max_mark    student.mark%TYPE;
   l_min_mark    student.mark%TYPE;
BEGIN
   -- get credit limits
   SELECT
      MIN(mark),
      MAX(mark),
      AVG(mark)
   INTO
      l_min_mark,
      l_max_mark,
      l_average_mark
   FROM student;
   SELECT
      name
   INTO
      l_name
   FROM
      student
WHERE
id=1;

   -- show the credits
   dbms_output.put_line('Min mark: ' || l_min_mark);
   dbms_output.put_line('Max mark: ' || l_max_mark);
   dbms_output.put_line('Avg mark: ' || l_average_mark);

   -- show customer credit
   dbms_output.put_line('student name: ' || l_name);
END;
```

Comments

PL/SQL **comments** allow you to describe the purpose of a line or a block of PL/SQL code.

When compiling the PL/SQL code, the Oracle precompiler ignores comments.

However, you should always use comments to make your code more readable and to help you and other developers understand it better in the future.

Comments

PL/SQL has two comment styles: **single-line** and **multi-line** comments:

- **Single-line comments:** A single-line comment starts with a double hyphen (**--**) that can appear anywhere on a line and extends to the end of the line.

```
-- valued added tax 10%
```

- **Multi-line comments:** A multi-line comment starts with a slash-asterisk (**/***) and ends with an asterisk-slash (***/**), and can span multiple lines.

```
/*
    This is a multi-line commet
    that can span multiple lines
*/
```

شركة تحالف الإمارات للحلول التقنية ذ.م.م.
TAHALUF AL EMARAT TECHNICAL SOLUTIONS L.L.C.

Chapter 08

IF Statement

The **IF statement** allows you to either execute or skip a sequence of statements, depending on a condition. The IF statement has the three forms:

- IF THEN
- IF THEN ELSE
- IF THEN ELSIF

IF Statement

IF THEN statement:

The condition is a Boolean expression that always evaluates to TRUE, FALSE, or NULL.

If the condition evaluates to TRUE, the statements after the THEN execute. Otherwise, the IF statement does nothing.

IF Statement

Syntax :

```
IF condition THEN
    statements;
END IF;
```

IF Statement

Example :

```
DECLARE n_sales NUMBER := 2000000;
BEGIN
   IF n_sales > 100000 THEN
      DBMS_OUTPUT.PUT_LINE( 'Sales revenue is greater 
than 100K ' );
   END IF;
END;
```

IF Statement

شركة تحالف الإمارات للحلول التقنية ذ.م.م.
TAHALUF AL EMARAT TECHNICAL SOLUTIONS L.L.C.

IF THEN ELSE statement:

The **IF THEN ELSE** statement has the following structure:

```
IF condition THEN
   statements;
ELSE
   else_statements;
END IF;
```

IF Statement

Example :

```
DECLARE
  n_sales NUMBER := 300000;
  n_commission NUMBER( 10, 2 ) := 0;
BEGIN
  IF n_sales > 200000 THEN
    n_commission := n_sales * 0.1;
  ELSE
    n_commission := n_sales * 0.05;
  END IF;
END;
```

IF Statement

TAHALUF AL EMARAT TECHNICAL SOLUTIONS L.L.C.

IF THEN ELSIF statement:

The following illustrates the structure of the **IF THEN ELSIF** statement:

```
IF condition_1 THEN
  statements_1
ELSIF condition_2 THEN
  statements_2
[ ELSE
    else_statements
]
END IF;
```

IF Statement

Example :

```
DECLARE
  n_sales NUMBER := 300000;
  n_commission NUMBER( 10, 2 ) := 0;
BEGIN
  IF n_sales > 200000 THEN
    n_commission := n_sales * 0.1;
  ELSIF n_sales <= 200000 AND n_sales > 100000 THEN
    n_commission := n_sales * 0.05;
  ELSIF n_sales <= 100000 AND n_sales > 50000 THEN
    n_commission := n_sales * 0.03;
  ELSE
    n_commission := n_sales * 0.02;
  END IF;
END;
```

IF Statement

Nested IF statement:

You can nest an IF statement within another IF statement as shown below:

```
IF condition_1 THEN
    IF condition_2 THEN
        nested_if_statements;
    END IF;
ELSE
    else_statements;
END IF;
```

CASE Statement

The **CASE** statement chooses one sequence of statements to execute out of many possible sequences.

The CASE statement has two types: **simple CASE statement** and **searched CASE statement**.

Both types of the CASE statements support an optional ELSE clause.

CASE Statement

شركـــة تحالـف الإمـــارات للحـلـــول التـقـنـيـة ذ.م.م.
TAHALUF AL EMARAT TECHNICAL SOLUTIONS L.L.C.

Simple CASE statement:
A simple CASE statement evaluates a single expression and compares the result with some values.

The simple CASE statement has the following structure:

```
CASE selector
WHEN selector_value_1 THEN
   statements_1
WHEN selector_value_1 THEN
   statement_2
ELSE
   else_statements
END CASE;
```

CASE Statement

```
DECLARE
 c_grade CHAR( 1 );
 c_rank  VARCHAR2( 20 );
BEGIN
 c_grade := 'B';
 CASE c_grade
 WHEN 'A' THEN
  c_rank := 'Excellent' ;
 WHEN 'B' THEN
  c_rank := 'Very Good' ;
 WHEN 'C' THEN
  c_rank := 'Good' ;
 WHEN 'D' THEN
  c_rank := 'Fair' ;
 WHEN 'F' THEN
  c_rank := 'Poor' ;
 ELSE
  c_rank := 'No such grade' ;
 END CASE;
 DBMS_OUTPUT.PUT_LINE( c_rank );
END;
```

CASE Statement

Searched CASE statement:

The searched CASE statement evaluates multiple Boolean expressions and executes the sequence of statements associated with the first condition that evaluates to TRUE.

CASE Statement

The searched CASE statement has the following structure:

```
CASE
WHEN condition_1 THEN statements_1
WHEN condition_2 THEN statements_2
...
WHEN condition_n THEN statements_n
[ ELSE
  else_statements ]
END CASE;]
```

CASE Statement

```
DECLARE
  n_sales      NUMBER;
  n_commission NUMBER;
BEGIN
  n_sales := 150000;
  CASE
  WHEN n_sales    > 200000 THEN
    n_commission := 0.2;
  WHEN n_sales   >= 100000 AND n_sales < 200000 THEN
    n_commission := 0.15;
  WHEN n_sales   >= 50000 AND n_sales < 100000 THEN
    n_commission := 0.1;
  WHEN n_sales    > 30000 THEN
    n_commission := 0.05;
  ELSE
    n_commission := 0;
  END CASE;

  DBMS_OUTPUT.PUT_LINE( 'Commission is ' ||
n_commission * 100 || '%'
  );
END;
```

شركة تحالف الإمارات للحلول التقنية ذ.م.م.
TAHALUF AL EMARAT TECHNICAL SOLUTIONS L.L.C.

GOTO statement

The **GOTO** statement allows you to transfer control to a labeled block or statement. The following illustrates the syntax of the GOTO statement:

```
GOTO label_name;
```

The label_name is the name of a label that identifies the target statement. In the program, you surround the label name with double enclosing angle brackets as shown below:

```
<<label_name>>;
```

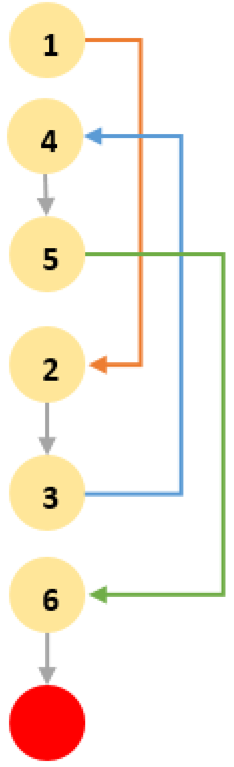
GOTO statement

```
BEGIN
   GOTO second_message;

  <first_message>
   DBMS_OUTPUT.PUT_LINE('Hello');
   GOTO the_end;

   <second_message>;
   DBMS_OUTPUT.PUT_LINE('PL/SQL GOTO Demo');
   GOTO first_message;

   <the_end>
   DBMS_OUTPUT.PUT_LINE('and good bye...');
END;
```



GOTO statement

شركــة تحالــف الإمـــارات للحــلــــول التـقـنـيــة ذ.م.م.
TAHALUF AL EMARAT TECHNICAL SOLUTIONS L.L.C.

```
BEGIN
  GOTO second_message;

  <<first_message>>
  DBMS_OUTPUT.PUT_LINE( 'Hello' );
  GOTO the_end;

  <<second_message>>
  DBMS_OUTPUT.PUT_LINE( 'PL/SQL GOTO
Demo' );
  GOTO first_message;

  <<the_end>>
  DBMS_OUTPUT.PUT_LINE( 'and good bye...' );

END;
```

Iterative processing with loops

Nested IF statement:

You can nest an IF statement within another IF statement as shown below:

```
<<label>> LOOP
   statements;
END LOOP loop_label;
```

Iterative processing with loops

```
DECLARE
 l_counter NUMBER := 0;
BEGIN
 LOOP
  l_counter := l_counter + 1;
  IF l_counter > 3 THEN
   EXIT;
  END IF;
  dbms_output.put_line( 'Inside loop: ' || l_counter )  ;
 END LOOP;
 -- control resumes here after EXIT
 dbms_output.put_line( 'After loop: ' || l_counter );
END;
```