# CHAPTER 6

1. **Variables**

* Use of Variables

Variables can be used for:

* + Temporary storage of data
  + Manipulation of stored values
  + Reusability
* Syntax:

***identifier* [CONSTANT] *datatype* [NOT NULL]**

**[:= | DEFAULT *expr*];**

In the syntax:

*identifier* Is the name of the variable

CONSTANT Constrains the variable so that its value cannot change (Constants must be initialized.)

*data type* Is a scalar, composite, reference, or LOB data type (This course covers only scalar, composite, and LOB data types.)

NOT NULL Constrains the variable so that it contains a value (NOT NULL

variables must be initialized.)

*expr* Is any PL/SQL expression that can be a literal expression, another variable, or an expression involving operators and functions

* Variable Name:
  + Must start with a letter
  + Can include letters or numbers
  + Can include special characters (such as $, \_, and #)
  + Must contain no more than 30 characters
  + Must not include reserved words
* Example:

**DECLARE**

v\_myName **VARCHAR2**(20);

**BEGIN**

DBMS\_OUTPUT.PUT\_LINE('My name is: '|| v\_myName);

v\_myName := 'John';

DBMS\_OUTPUT.PUT\_LINE('My name is: '|| v\_myName);

**END**;

**DECLARE**

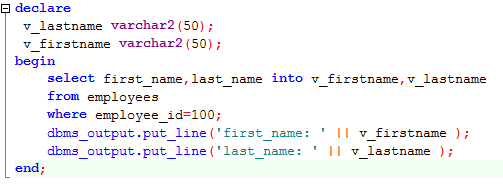
v\_myName **VARCHAR2**(20):= 'John';

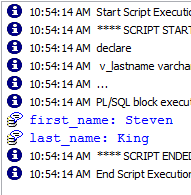
**BEGIN**

v\_myName := 'Steven';

DBMS\_OUTPUT.PUT\_LINE('My name is: '|| v\_myName);

**END**;





\* Basic

**DECLARE**

birthday **DATE**;

emp\_count **SMALLINT** := 0;

\* Constants

**DECLARE**

credit\_limit **CONSTANT** **REAL** := 5000.00;

max\_days\_in\_year **CONSTANT** **INTEGER** := 366;

urban\_legend **CONSTANT** **BOOLEAN** := **FALSE**;

\* Using Default

blood\_type **CHAR** := 'O';

can be rewritten **as** **follows**:

blood\_type **CHAR** **DEFAULT** 'O';

\* Using Not Null

**DECLARE**

acct\_id **INTEGER**(4) **NOT** **NULL** := 9999;

\* Using the %TYPE Attribute

The %TYPE attribute provides the datatype of a variable or database column.

• Is used to declare a variable according to:

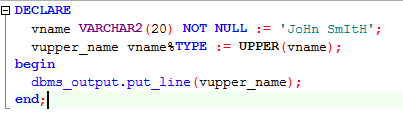
– A database column definition

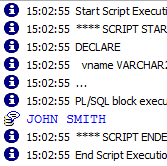
– Another declared variable

• Is prefixed with:

– The database table and column name

– The name of the declared variable





\* Using the %ROWTYPE Attribute

The %ROWTYPE attribute provides a record type that represents a row in a table or view. Columns in a row and corresponding fields in a record have the same names and datatypes. However, fields in a %ROWTYPE record do not inherit constraints, such as the NOT NULL or check constraint, or default values

**DECLARE**

emprec employees%**ROWTYPE**;

**BEGIN**

emprec.EMPLOYEE\_ID := **NULL**; -- this works, null constraint is not inherited

emprec.FIRST\_NAME := 'test';

DBMS\_OUTPUT.PUT\_LINE('emprec.deptname: ' || emprec.FIRST\_NAME);

**END**;

1. **Control Structures**
   1. **Conditional Control**
      1. If statement

Syntax:

**IF *condition* THEN**

***statements*;**

**[ELSIF *condition* THEN**

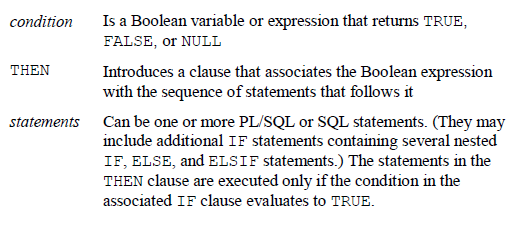
***statements*;]**

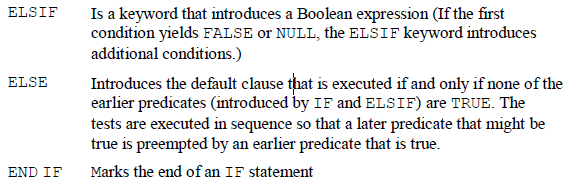
**[ELSE**

***statements*;]**

**END IF;**

In the syntax:





Example:

**DECLARE**

v\_myage **number** :=10;

**BEGIN**

**IF** v\_myage < 11 **THEN**

DBMS\_OUTPUT.PUT\_LINE(' I am a child ');

**ELSE**

DBMS\_OUTPUT.PUT\_LINE(' I am not a child ');

**end** **if**;

**END**;

* + 1. Case when statement

Syntax:

**CASE selector**

**WHEN expression1 THEN result1**

**WHEN expression2 THEN result2**

**...**

**WHEN expressionN THEN resultN**

**[ELSE resultN+1]**

**END;**

**Example:**

**SELECT** last\_name, job\_id, salary,

**CASE** job\_id **WHEN** 'IT\_PROG' **THEN** 1.10\*salary

**WHEN** 'ST\_CLERK' **THEN** 1.15\*salary

**WHEN** 'SA\_REP' **THEN** 1.20\*salary

**ELSE** salary

**END** "REVISED\_SALARY"

**FROM** employees;

**SELECT** last\_name,salary,

(**CASE** **WHEN** salary<5000 **THEN** 'Low'

**WHEN** salary<10000 **THEN** 'Medium'

**WHEN** salary<20000 **THEN** 'Good'

**ELSE** 'Excellent'

**END**) qualified\_salary

**FROM** employees;

* + 1. Decode function

Syntax:

**DECODE(*col|expression, search1, result1***

**[*, search2, result2,...,*][*, default*])**

**Example:**

**SELECT** last\_name, job\_id, salary,

**DECODE**(job\_id, 'IT\_PROG', 1.10\*salary,'ST\_CLERK', 1.15\*salary,

'SA\_REP', 1.20\*salary,salary) REVISED\_SALARY

**FROM** employees;

* 1. Interactive Control
     1. Loop statement

Loops repeat a statement (or a sequence of statements) multiple times.

There are three loop types:

– Basic loop

– FOR loop

– WHILE loop

* Basic loop
  + Syntax:

**LOOP**

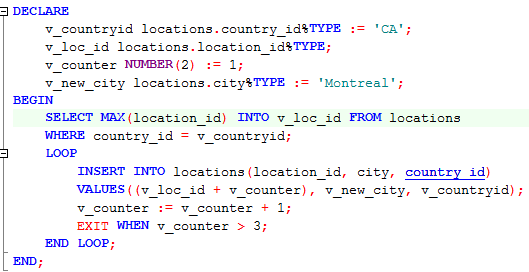
***statement1*;**

**. . .**

**EXIT [WHEN *condition*];**

**END LOOP;**

* + Example



* For loop
  + Syntax

**FOR *counter* IN [REVERSE]**

***lower\_bound..upper\_bound* LOOP**

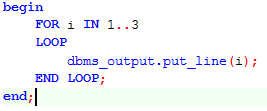
**statement1;**

**statement2;**

**. . .**

**END LOOP;**

* + Example:



* While loop
  + Syntax:

**WHILE *condition* LOOP**

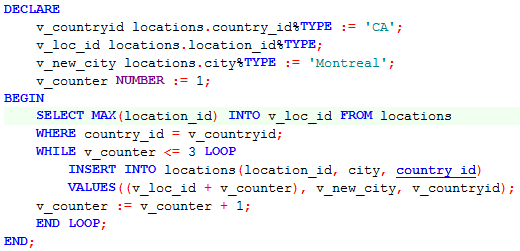
**statement1;**

**statement2;**

**. . .**

**END LOOP;**

* + Example:



* 1. **Sequential Control**
     1. **Goto**

The GOTO statement branches unconditionally to a statement label or block label. The label must be unique within its scope and must precede an executable statement or a PL/SQL block. The GOTO statement transfers control to the labelled statement or block.

Example:

**DECLARE**

p **VARCHAR2**(30);

n **PLS\_INTEGER** := 37; -- test any integer > 2 for prime

**BEGIN**

**FOR** j **in** 2..**ROUND**(**SQRT**(n)) **LOOP**

**IF** n **MOD** j = 0 **THEN** -- test for prime

p := ' is not a prime number'; -- not a prime number

**GOTO** print\_now;

**END** **IF**;

**END** **LOOP**;

p := ' is a prime number';

<<print\_now>>

DBMS\_OUTPUT.PUT\_LINE(**TO\_CHAR**(n) || p);

**END**;

* + 1. **Null statements**

The NULL statement only passes control to the next statement. Some languages refer to such an instruction as a no-op (no operation).

1. **Cursor**

A cursor is a pointer to the private memory area allocated by the Oracle Server. It is used to handle the result set of a SELECT statement.

There are two types of cursors: implicit and explicit.

– Implicit: Created and managed internally by the Oracle

Server to process SQL statements

– Explicit: Declared explicitly by the programmer

Example

Implicit

**DECLARE**

l\_vc\_first\_name employees.first\_name%**type**;

l\_vc\_last\_name employees.last\_name%**type**;

**BEGIN**

**SELECT** first\_name, last\_name

**INTO** l\_vc\_first\_name, l\_vc\_last\_name

**FROM** employees

**WHERE** employee\_id=100;

dbms\_output.put\_line('Number of rows processed: '||**sql**%rowcount);

**END**;

**DECLARE**

l\_vc\_first\_name employees.first\_name%**type**;

l\_vc\_last\_name employees.last\_name%**type**;

**BEGIN**

**for** cur **in** (**SELECT** first\_name, last\_name **FROM** employees)

**loop**

dbms\_output.put\_line(cur.first\_name);

**end** **loop**;

**END**;

**DECLARE**

**CURSOR** cur **IS** **SELECT** first\_name, last\_name **FROM** employees;

**BEGIN**

**FOR** loop\_emp **IN** cur

**LOOP**

dbms\_output.put\_line(loop\_emp.first\_name);

dbms\_output.put\_line('Number of rows processed: ||**NVL**(**TO\_CHAR**(cur%rowcount),'Null'));

**END** **LOOP** loop\_emp;

**END**;

Explicit:

**DECLARE**

emp\_rec employees%**rowtype**;

**CURSOR** emp\_cur **IS** **SELECT** \* **FROM** employees;

**BEGIN**

**OPEN** emp\_cur;

**LOOP**

EXIT **WHEN** emp\_cur%**NOTFOUND**;

**FETCH** emp\_cur **INTO** emp\_rec;

dbms\_output.put\_line (emp\_rec.first\_name || ' ' || emp\_rec.last\_name);

**end** **loop**;

**CLOSE** emp\_cur;

**END**;

1. **Handing Exceptions**

An exception is a PL/SQL error that is raised during program execution

Syntax:

**EXCEPTION**

**WHEN *exception1* [OR *exception2* . . .] THEN**

***statement1*;**

***statement2*;**

**. . .**

**[WHEN *exception3* [OR *exception4* . . .] THEN**

***statement1*;**

***statement2*;**

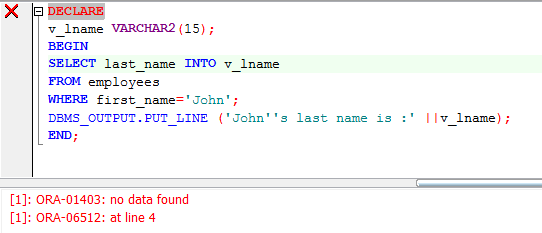
**. . .]**

**[WHEN OTHERS THEN**

***statement1*;**

***statement2*;**

Example:



**DECLARE**

v\_lname **VARCHAR2**(15);

**BEGIN**

**SELECT** last\_name **INTO** v\_lname

**FROM** employees

**WHERE** first\_name='John';

DBMS\_OUTPUT.PUT\_LINE ('John''s last name is :' ||v\_lname);

**EXCEPTION**

**WHEN** NO\_DATA\_FOUND **THEN**

DBMS\_OUTPUT.PUT\_LINE (' Your select statement retrieved 0 row.');

**END**;

**DECLARE**

v\_deptno **NUMBER** := 500;

v\_name **VARCHAR2**(20) := 'Testing';

e\_invalid\_department **EXCEPTION**;

**BEGIN**

**UPDATE** departments

SET department\_name = v\_name

**WHERE** department\_id = v\_deptno;

**IF** **SQL**%**NOTFOUND** **THEN**

**RAISE** e\_invalid\_department;

**END** **IF**;

**COMMIT**;

**EXCEPTION**

**WHEN** e\_invalid\_department **THEN**

DBMS\_OUTPUT.PUT\_LINE('No such department id.');

**END**;

1. **Practices**
   1. The %TYPE attribute:

1. Is used to declare a variable according to a database column definition

2. Is used to declare a variable according to a collection of columns in a database table or view

3. Is used to declare a variable according to the definition of another declared variable

4. Is prefixed with the database table and column name or the name of the declared variable

* 1. Using the DECODE function, write a query that displays the grade of all employees based on the value of the column JOB\_ID, using the following data:

***Job Grade***

AD\_PRES 🡪A

ST\_MAN 🡪 B

IT\_PROG 🡪 C

SA\_REP 🡪 D

ST\_CLERK 🡪 E

None of the above 🡪 0

* 1. Rewrite the statement in the preceding exercise using the CASE syntax.
  2. Display the last name, hire date, and day of the week on which the employee started. Label the column DAY. Order the results by the day of the week, starting with Monday.



* 1. The HR department needs a report to display the employee number, last name, salary, and salary increased by 15.5% (expressed as a whole number) for each employee. Label the column New Salary
  2. Write a query that displays the last name (with the first letter uppercase and all other letters lowercase) and the length of the last name for all employees whose name starts with the letters *J, A,* or *M.*
  3. Create table Emp3 table based on the structure of the Employees table. Use cursor to insert data from Employees to Emp3