PROGRAMACIÓN DE BASE DE DATOS
PL/SQL

Valeria Beratto Ulloa - vberatto@ubiobio.cl

OBJETIVO DE ESTA SESIÓN

- Conocer estructuras de programas en PL/SQL, tipos de datos y funciones en Oracle 18c
- Conocer sintaxis de sentencias de control en PL/SQL (IF, CASE, LOOP, FOR y WHILE)
- Ejecutar programas en PL/SQL con y sin sentencias en SQL.

INTRODUCCIÓN A PL/SQL

```
CREATE TABLE prueba(nro NUMBER(3) PRIMARY KEY,
dato VARCHAR2(80));

BEGIN
FOR i IN 1..500 LOOP
   INSERT INTO prueba
   VALUES (i,  'Comenzando');
END LOOP;
END;
```

BLOQUES PL/SQL

```
DECLARE
Sección de declaración
BEGIN
Sección ejecutable
EXCEPTION
Sección de manejo de excepciones
END;
```

- Las secciones de manejo de excepciones y de declaración son opcionales.
- Los bloques pueden contener otros bloques (sub-bloques)
- Los comentarios van entre /* */. Si no ocupan más de una línea, se pueden escribir después de -- (dos guiones).

VARIABLES Y CONSTANTES

- Tipos de datos* en PL/SQL:

NUMBER, CHAR, VARCHAR2, DATE, BOOLEAN, entre otros.

Más información tipos de datos

- La sintaxis para declarar variables o constantes es:

```
nombre [CONSTANT] TIPO [NOT NULL][:= expresión];
```

Los corchetes indican las partes opcionales

- No se diferencian mayúsculas y minúsculas (No es Case Sensitive)
- NUMBER, CHAR,VARCHAR2 tienen precisión.
- El operador de asignación es **:=** y el de igualdad es **= .**

ALCANCE DE VARIABLES O CONSTANTES

El alcance o visibilidad de las variables sigue estas reglas:

1. Una variable es visible en el bloque en el cual se declara y en todos sus sub-bloques, a menos que se aplique la regla 2.
2. Si se declara una variable en un sub-bloque con el mismo nombre que una variable del bloque contenedor (externo), la variable del sub-bloque es la que tiene prioridad en el sub-bloque*.

* Es posible acceder en el sub-bloque a la variable del bloque contenedor (externo) mediante etiquetas (luego se ejemplifican), pero lo más sencillo es usar nombres diferentes para las variables.

ALCANCE DE VARIABLES

```
DECLARE
a NUMBER(2);
BEGIN
 a := &Ingrese_valora;
 DBMS_OUTPUT.PUT_LINE('Valor de a externa '|| a);
   DECLARE
   a NUMBER(3) := 20;
   BEGIN
   DBMS_OUTPUT.PUT_LINE('Valor de a interna '|| a);
   END;
 DBMS_OUTPUT.PUT_LINE('Valor de a '|| a);
END;
```

Ingreso de dato por teclado

Operador de concatenación ||

Sub-bloque

Para imprimir en pantalla *

* Se debe ejecutar primero SET SERVEROUTPUT ON;

DDL Y DML EN PL/SQL

- En PL/SQL se puede usar directamente el sublenguaje de manipulación de datos **DML** de SQL, es decir, INSERT, DELETE, UPDATE y SELECT (el SELECT requiere usar INTO o estar asociado con un cursor, ver luego).
- Para usar sentencias **DDL** en PL/SQL, es decir, **CREATE, DROP, ALTER** se puede hacer así:

```
BEGIN
EXECUTE IMMEDIATE 'CREATE TABLE prueba(nro
NUMBER(3) PRIMARY KEY, dato VARCHAR2(80))';
END;
```

DDL Y DML EN PL/SQL

- La sentencia DDL **NO** lleva punto y coma dentro de las comillas simples.
- Lo que sigue a `IMMEDIATE` puede ser una variable de caracteres

```
BEGIN
EXECUTE IMMEDIATE 'INSERT INTO prueba VALUES(1,
 'hola' )';
END;
```

- Es más simple:

```
BEGIN
INSERT INTO prueba VALUES(1,  'hola' );
END;
```

FUNCIONES

- En PL/SQL, las funciones numéricas (SQRT, ROUND, POWER, etc.), de caracteres (LENGTH, UPPER, INITCAP, SUBSTR, etc.) , de fechas (ADD_MONTHS, MONTHS_BETWEEN, etc.); se pueden usar por fuera de una sentencia SQL pero las funciones de grupo o agregación (COUNT, SUM, AVG, MAX, etc.) sólo se pueden usar dentro de una sentencia SQL.

- Más información de funciones

EJEMPLO

```
CREATE TABLE emp(
 cod NUMBER(8) PRIMARY KEY,
 nom VARCHAR2(8) NOT NULL,
 fecha_ing DATE,
 sueldo NUMBER(8) CHECK(sueldo > 0)
);
```

EJEMPLO

```
DECLARE
fi  emp.fecha_ing%TYPE;
nom VARCHAR2(20)  := INITCAP( 'paz ortiz');
BEGIN
fi  := ADD_MONTHS(SYSDATE,-14);
INSERT INTO emp
VALUES (4329,SUBSTR(nom,1,8),fi,10000);
END;
```

fi hereda el tipo de dato fecha_ing de emp

SUBSTR(texto, *posicion, longitud_subchar*)

Acá también se pueden colocar los valores directamente y prescindir de las variables.

CONSULTA SQL EN PL/SQL

- Se debe proporcionar un "lugar" para guardar los datos devueltos por una consulta (SELECT)
- Esto se puede lograr mediante la cláusula SELECT … INTO.
- Sin embargo, un SELECT . . . INTO debe retornar **una y solo una** fila:
  - Si la consulta no recupera filas o recupera múltiples filas, ocurre un error (**excepción,** se verán luego).
  - Los cursores (**se verá más adelante**) sirven para consultas que recuperan 0, 1 o más filas.

EJEMPLO 2

```
DECLARE
 nom emp.nom%TYPE;
 sue emp.sueldo%TYPE;
 cuantos NUMBER(8);
BEGIN
 SELECT nom, sueldo INTO nom, sue
 FROM emp WHERE cod = 4329;
 DBMS_OUTPUT.PUT_LINE('El empleado ' || nom || ' tiene sueldo ' ||
  sue);
 SELECT COUNT(*) INTO cuantos
 FROM emp;
DBMS_OUTPUT.PUT_LINE('Total empleados ' || cuantos);
END;
```

CONTROL DE FLUJO

- Las comparaciones lógicas son la base del control condicional en PL/SQL.

  Los resultados de las comparaciones son verdadero (TRUE), falso (FALSE) o nulo (NULL).

- Cualquier "cosa" comparada con NULL retorna NULL (desconocido).

- Los operadores lógicos son : >, <, =, !=, <=, >=, <>

IF

La sentencia IF tiene la sintaxis:

```
IF condición THEN
 secuencia de instrucciones
[ELSIF condición THEN
 secuencia de instrucciones]
--Los ELSIF se pueden repetir
[ELSE
 secuencia de instrucciones]
END IF;
```

EJEMPLO IF

```
DECLARE
 a NUMBER := NULL;
BEGIN
 IF a = a THEN
  DBMS_OUTPUT.PUT_LINE('O sea que NULL = NULL');
 ELSIF a <> a THEN
  DBMS_OUTPUT.PUT_LINE('O sea que NULL <> NULL');
 ELSE
  DBMS_OUTPUT.PUT_LINE('Indefinido, NULL no es ni = ni <> a NULL');
 END IF;
END;
```

Comparación con nulo: ¿Qué imprime el siguiente programa?:

* Se debe ejecutar primero `SET SERVEROUTPUT ON;`

USO CASE

Lo anterior también se puede escribir con CASE así:

```
DECLARE
 a NUMBER := NULL;
BEGIN
CASE
WHEN a = a THEN
  DBMS_OUTPUT.PUT_LINE('O sea que NULL = NULL');
WHEN a <> a THEN
  DBMS_OUTPUT.PUT_LINE('O sea que NULL <> NULL');
 ELSE
  DBMS_OUTPUT.PUT_LINE('Indefinido, NULL no es ni = ni <> a NULL');
 END CASE;
END;
```

CICLOS O ITERACIONES

a) Ciclo simple sin límite: LOOP

```
LOOP
 secuencia de instrucciones
END LOOP;
```

Para salir del ciclo se usa:

```
EXIT [WHEN condición];
```

EJEMPLO LOOP

```
DECLARE
 cont NUMBER(4) := 0;
BEGIN
DELETE prueba;
 LOOP
        INSERT INTO prueba VALUES(cont,  'Usando LOOP'||
        CEIL(DBMS_RANDOM.VALUE(1,100000)));
        cont := cont + 1;
  EXIT WHEN cont = 1000;
 END LOOP;
END;
```

CEIL(n): Redondea n hasta el valor superior.

DBMS_RANDOM: Asigna valores aleatorios

- .VALUE
- .STRING

CICLO PARA: FOR

Permite repetir una secuencia de instrucciones un número fijo de veces. Su sintaxis es:

```
FOR índice IN [REVERSE] entero .. entero LOOP
 secuencia de instrucciones
END LOOP;
```

Notas: - El incremento del FOR siempre es 1.

- Aunque el ciclo se haga "en reversa" los límites siempre se colocan de menor a mayor. Veamos un ejemplo:

EJEMPLO FOR

```
BEGIN
DELETE PRUEBA;
FOR i IN REVERSE 1..500 LOOP
   INSERT INTO prueba
   VALUES (i,  'Aprendiendo FOR');
 END LOOP;
END;
```

CICLO MIENTRAS QUE: WHILE

WHILE repetirá una secuencia de instrucciones hasta que la condición controladora del ciclo deje de ser cierta. Su sintaxis es:

```
WHILE condición LOOP
 secuencia de instrucciones
END LOOP;
```

EJEMPLO WHILE

```
DECLARE
  cont NUMBER(3) := 500;
BEGIN
WHILE cont > 0 LOOP
   INSERT INTO PRUEBA VALUES
   (cont, DBMS_RANDOM.STRING('u',60) || cont);
   cont := cont - 1;
  END LOOP;
END;
```

u: mayúsculas
l: minúsculas
a: combinación de mayúsculas y minúsculas
x: alfanuméricos

Tamaño

EJERCICIOS DE TRABAJO AUTÓNOMO

- Ejecute los códigos que contiene esta presentación.
- Revise los link de los tipos de datos y las funciones disponibles de Oracle en su documentación.
- Próxima clases realizaremos ejercicios en conjunto