Facultad de Estudios Superiores Acatlán

# PLSQL

José Antonio Coria Fernández
Email:
diplomadobd@gmail.com

# Contenido



- Introducción
- Características
- Caracteres permitidos, variables y tipos
- Tipo de Datos
- Operadores
- Estructuras de Control
- Cursores
- Ejercicios

# Introducción



PL/SQL es un lenguaje procedural (basado en el lenguaje ADA) diseñado por ORACLE.

Incorpora características propias de los lenguajes de programación como son:

- La declaración de variables y constantes,
- Estructuras de control
- Manejo de excepciones
- Estructura modular (procedimientos y funciones).

# Introducción



- Los programas creados con PL/SQL se pueden almacenar en la base de datos como cualquier objeto de ésta
- En la base de datos Oracle 11*g*, *PL/SQL evolucionó* de un lenguaje interpretado a un lenguaje compilado nativamente

# Característica



- La estructura básica “característica” del lenguaje se denomina bloque
- Un bloque de PL/SQL puede dividirse en dos grupos:
  anónimos y con nombre.
- Los bloques anónimos, no son almacenados en la base de datos y no pueden ser referenciados posteriormente.
- Por el contrario, los bloques con nombre, si se almacenan en la base de datos y se referencian vía el nombre que les fue asignado y son utilizados cuando se crean procedimientos, funciones y paquetes.

# Característica



- Un bloque se compone de tres partes:

```
[DECLARE
   (opcional)
  --- declaraciones ]
BEGIN
   (obligatorio)
 --- Instrucciones ejecutables
[EXCEPTION
   (opcional)
 --- Rutinas de manejo de excepciones]
END;
   (oligatorio)
/
```

# Característica



- La estructura del bloque anterior, tiene tres zonas claramente definidas:
  - Declaraciones
  - Instrucciones
  - Excepciones

# Características



- Declaraciones: Contiene la definición y posiblemente la inicialización de objetos locales tales como variables, constantes, cursores y excepciones.
- Instrucciones: se manipulan las instrucciones
- Excepciones: maneja cualquier excepción que produzca en la ejecución.

- Es posible anidar sub-bloques en la sección de instrucciones y de excepciones, pero no en la sección de declaraciones

# Características



- Ejemplo de la Sección de declaración

```
DECLARE
    var_nombre VARCHAR2(35);
    var_apellido VARCHAR2(35);
      Vrent NUMBER(6,2) NOT NULL:=600;
    contador CONSTANT NUMBER :=0;
```

Nombre de la variable
Tipo de dato y tamaño
Terminación de cada declaración con [;]

# Características



- Ejemplo de la Sección de Instrucciones

```
BEGIN
   SELECT nombre, apellido
        INTO var_nombre, var_apellido
FROM estudiantes
WHERE idestudiante=123;
DBMS_OUTPUT.PUT_LINE ('Nombre estudiante: '||
   var_nombre||' '||var_apellido);
END;
/
```

# Características



- Ejemplo de la Sección de Excepciones – contiene las declaraciones que se ejecutan cuando un error en tiempo de ejecución ocurre

```
EXCEPTION
   WHEN NO DATA_FOUND THEN
   DBMS_OUTPUT.PUT_LINE (‘No existe un estudiante
   con’ || ‘id de estudiante 123’);
END;
/
```

# Ejemplo completo



```
DECLARE
      var_nombre VARCHAR2(35);
      var_apellido VARCHAR2(35);
    BEGIN
       SELECT nombre, apellido
       INTO var_nombre, var_apellido
       FROM estudiante
      WHERE idestudiante = 123;
      DBMS_OUTPUT.PUT_LINE (‘Nombre estudiante: '||var_nombre||' '||
      var_apellido);
    EXCEPTION
      WHEN NO_DATA_FOUND THEN
        DBMS_OUTPUT.PUT_LINE (No existe un estudiante con '||
        ‘idestudiante 123');
END;
/
```

# Ejercicio:



- Introduzca el siguiente código:

```
SET SERVEROUTPUT ON;
    DECLARE
        v_cadena varchar2(20):='Hola Mundo';
    BEGIN
        dbms_output.put_line(v_cadena);
END;
/
```

# Caracteres permitidos, variables y tipos



- Al programar en PL/SQL sólo pueden emplearse los siguientes caracteres:
  - todas las letras mayúsculas y minúsculas; todos los dígitos de 0 a 9;
  - los símbolos:

( ) + -* / < > = ! ; : . ’

@ % , ” # $ & _ |

{ } ? [ ] ^ ~.

# Caracteres permitidos, variables y tipos



- Con los caracteres permitidos, se construyen los *identificadores* con los cuales se da nombre a los objetos PL/SQL, tales como variables, cursores, tipos y subprogramas.
- Al construir los identificadores se deben tomar en cuenta las siguientes reglas:
  - Deben comenzar por una letra.
  - No deben exceder de 30 caracteres, siendo todos significativos.

# Caracteres permitidos, variables y tipos



- No deben incluir espacios.
- No deben ser iguales a las palabras reservadas de PL/SQL (aquellas que tienen un significado especial para el lenguaje).

*Cabe señalar que PL/SQL no distingue entre mayúsculas y minúsculas, por lo que ambas formas de letra son equivalentes, excepto en el interior de una cadena de caracteres delimitada por comillas.*

# Caracteres permitidos, variables y tipos



Introduzca el siguiente código:

```
set serveroutput on size 5000;
declare
   exception varchar2(15);
Begin
   exception:='Esto es un ejercicio para no usar
                  palabras reservadas';
   dbms_output.put_line(exception);
End;
/
```

# Tipo de Datos



- Casi todos los tipos de datos manejados por PL/SQL son similares a los soportados por SQL. A continuación se muestran los TIPOS de DATOS más comunes:
- **NUMBER** (*Numérico*): Almacena números enteros o de punto flotante, virtualmente de cualquier longitud, aunque puede ser especificada la precisión (Número de digitos) y la escala ,que determina el número de decimales. Ejemplo:

saldo **NUMBER**(16,2);

/* Indica que puede almacenar un valor numérico de 16 posiciones, 2 de ellas decimales. Es decir, 14 enteros y dos decimales */

# Tipo de Datos



- **CHAR** (*Caracter*): Almacena datos de tipo caracter con una longitud máxima de 32767 y cuyo valor de longitud predeterminado es 1. Ejemplo:

  nombre **CHAR**(20);

/* Indica que puede almacenar valores alfanuméricos de 20 posiciones */

- **VARCHAR2** (*Caracter de longitud variable*): Almacena datos de tipo caracter empleando sólo la cantidad necesaria aún cuando la longitud máxima sea mayor.

# Tipo de Datos



nombre **VARCHAR2**(20);

/* Indica que puede almacenar valores alfanuméricos de hasta 20 posicones */

/* Cuando la longitud de los datos sea menor de 20 no se rellena con blancos */

- **BOOLEAN** (*lógico*): Se emplea para almacenar valores TRUE o FALSE.

  error **BOOLEAN NOT NULL := TRUE**;

# Tipo de Datos



- **DATE** (*Fecha*): Almacena datos de tipo fecha. Las fechas se almacenan internamente como datos numéricos, por lo que es posible realizar operaciones aritmeticas con ellas.

```
fecha DATE;
 fecha := ’01-JUN-07’;
 fecha:= SYSDATE +1;
```

/* Estos datos contienen dos valores, la fecha y la hora.*/
/* Almacena fechas desde el 1 de enero del 4712 a.C. hasta el 31 de diciembre del 4712 d.C. */

# Tipo de Datos



| Función | Devuelve | Ejemplo | Se visualiza |
|---|---|---|---|
| Y o YY o YYY | El o los dos o tres dígitos finales del año. | select to_char(sysdate, 'YYY') from dual; | 999 para todas las fechas en 1999. |
| SYEAR o YEAR | El año, utilizando S para incluir un signo menos para las fechas a.C. | select to char(sysdate, 'SYEAR') from dual; | -1112 en el año 1112 a. C. |
| Q | Trimestre del año (de enero a marzo = 1) | select to_char(sysdate, 'Q') from dual; | 2 para todas las fechas de junio |
| MM | Mes (0-12, diciembre=12) | select to_char(sysdate, 'MM') from dual; | 12 para fechas de diciembre |
| RM | Mes en números romanos | select to_char(sysdate, 'RM') from dual; | IV para todas las fechas de abril |
| Month | Nombre del mes con 9 caracteres | select to_char(sysdate, 'Month') from dual; | May seguido por 6 espacios para todas las fechas de mayo |
| WW | Semana del año | select to_char(sysdate, 'WW') from dual; | 24 el 13 de junio de 1998 |
| W | Semana del mes | slect to_char(sysdate, 'W') from dual | 1 el 1 de octubre de 1995 |
| DDD | Día del año: el 1 de enero es 001, el 1 de febrero es 032, etc. | select to_char(sysdate, 'DDD') from dual | 363 el 29 de diciembre de 1999 |
| DD | Día del mes | select to_char(sysdate, 'DD') from dual; | 04 el 4 de octubre de cualquier año |
| D | Día de la semana (1-7) | select to_char(sysdate, 'D') from dual | 1 el 14 de marzo de 1999 |
| DY | Abreviatura del nombre del día | select to_char(sysdate, 'DY') from dual | SUN el 28 de marzo de 1999 |
| HH o HH12 | La hora del día (1-12) | select to_char(sysdate, 'HH') from dual; | 02 si son las 2 y 8 minutos después de medianoche |
| HH24 | La hora del día utilizando un reloj de 24 horas | select to_char(sysdate, 'HH24') from dual; | 14 si son las 2 y 8 minutos del mediodía |
| MI | Minutos (0-59) | select to_char(sysdate, 'MI') from dual; | 17 si son las 4:17 de la tarde |
| SS | Segundos (0-59) | Select to_char(sysdate, 'SS') from dual; | 22 si son las 11:03:22 |

/* Formatos comunes para datos tipo fecha */

# Tipo de Datos



- **Atributos de tipo**. Un atributo de tipo PL/SQL es un modificador que puede ser usado para obtener información de un objeto de la base de datos. Ejemplos:
  - El atributo **%TYPE** permite conocer el tipo de una variable, constante o campo de la base de datos.
  - El atributo **%ROWTYPE** permite obtener los tipos de todos los campos de una tabla de la base de datos, de una vista o de un cursor.

# Tipo de Datos



- **Atributos de tipo**. Un atributo de tipo PL/SQL es un modificador que puede ser usado para obtener información de un objeto de la base de datos. Ejemplos:
  - El atributo **%TYPE** define el tipo de una variable utilizando una definición previa de otra variable o columna de la base de datos

```
DECLARE
 credito      REAL(7,2);
 debito       credito%TYPE;
```

# Tipo de Datos



- El atributo **%ROWTYPE** precisa el tipo de un registro (*record*) utilizando una definición previa de una tabla o vista de la base de datos. También se puede asociar a una variable como del tipo de la estructura retornada por un cursor . Ejemplo:

```
DECLARE
  empvar   emp%ROWTYPE;
```

En este ejemplo la variable *empvar* tomará el formato de un registro completo de la tabla *emp*

# Operadores en PL/SQL



- La siguiente tabla ilustra los operadores en

| Tipo de operador | Operadores |
|---|---|
| **Operador de asignación** | := (dos puntos + igual) |
| **Operadores aritméticos** | + (suma)<br>- (resta)<br>* (multiplicación)<br>/ (división)<br>** (exponente) |
| **Operadores relacionales o de comparación** | = (igual a)<br><> (distinto de)<br>< (menor que)<br>> (mayor que)<br>>= (mayor o igual a)<br><= (menor o igual a) |
| **Operadores lógicos** | AND (y lógico)<br>NOT (negacion)<br>OR (o lógico) |
| **Operador de concatenación** | \|\| |

# Operadores en PL/SQL



```
SET SERVEROUTPUT ON;
DECLARE
v_var1 NUMBER(5,3);
v_var2 NUMBER(5,3);
v_var3 NUMBER(5,3);
BEGIN
v_var1 := 11.234;
v_var2 := 12.345;
v_var3 := v_var1 + v_var2;
DBMS_OUTPUT.PUT_LINE('v_var1: '||v_var1);
DBMS_OUTPUT.PUT_LINE('v_var2: '||v_var2);
DBMS_OUTPUT.PUT_LINE('v_var3: '||v_var3);
END;
/
```

# Estructuras de Control



- Las estructuras de Control efectúan las siguientes tareas:

  - verifican una condición lógica
  - ramifican la ejecución de un programa.

# Estructuras de Control



- Estructura condicional **if**

```
IF (expresion) THEN
        -- Instrucciones
   ELSIF (expresion)  THEN
          -- Instrucciones
     ELSE
    -- Instrucciones
END IF;
```

# Estructuras de Control



- Ejercicio

```
DECLARE
-- Definir variable booleana
v_bool BOOLEAN;
BEGIN
-- La funcion NVL. Sustituye un valor cuando un valor nulo es
    encontrado
IF NOT NVL(v_bool,FALSE) THEN
dbms_output.put_line('Esto debe mostrarse');
ELSE
dbms_output.put_line('Esto no debe mostrarse');
END IF;
END;
/
```

# Estructuras de Control



- El bucle **LOOP,** se repite tantas veces como sea necesario hasta que se fuerza su salida con la instrucción **EXIT.** Su sintaxis es la siguiente:

```
LOOP
        -- Instrucciones
    IF (expresion) THEN
        -- Instrucciones
      EXIT;
  END IF;
END LOOP;
```

# Estructuras de Control



- Ejercicio

```
DECLARE
x NUMBER := 0;
BEGIN
LOOP – Depués de la declaración CONTINUE, el control continua aqui control
DBMS_OUTPUT.PUT_LINE ('Ciclo interno: x = ' || TO_CHAR(x));
x := x + 1;
CONTINUE WHEN x < 3; – Sentencia válida para oracle 11g
DBMS_OUTPUT.PUT_LINE
('Ciclo interno, despues de CONTINUE: x = ' || TO_CHAR(x));
EXIT WHEN x = 5;
END LOOP;
DBMS_OUTPUT.PUT_LINE ('Despues del Loop: x = ' || TO_CHAR(x));
END;
/
```

# Estructuras de Control



- El bucle **WHILE,** se repite mientras que se cumpla *expresion*.

**WHILE** (expresion) **LOOP**

-- Instrucciones

**END LOOP**;

# Estructuras de Control



- Ejercicio (1/2)

```
declare
  v_test varchar2(8) := 'SIGUE';
  n_numb number := 2;
  begin
  while v_test <> 'PARA' loop
  if n_numb > 5
  then v_test := 'PARA';
  end if;
  dbms_output.put_line (v_test||': '||n_numb);
  n_numb := n_numb + 1;
```

# Estructuras de Control



- Ejercicio (2/2)

```
 end loop;
v_test := 'ABAJO';
while n_numb > 1 AND v_test = 'ABAJO' loop
dbms_output.put_line (v_test||': '||n_numb);
n_numb := n_numb - 1;
end loop;
while 7 = 4 loop
NULL;  -- no entra aqui
end loop;
end;
```

# Estructuras de Control



- El bucle **FOR**, se repite tanta veces como le indiquemos en los identificadores *inicio* y *final.*

**FOR** contador **IN** [**REVERSE**] inicio..final
**LOOP**
    -- Instrucciones
**END LOOP**;

En el caso de especificar **REVERSE** el bucle se recorre en sentido inverso.

# Estructuras de Control



## Ejemplo:

```
BEGIN
FOR i IN 1..10 LOOP
dbms_output.put_line('El valor del indice es ['||i||']');
END LOOP;
END;
/
```

# Estructuras de Control



- La estructura CASE trabaja como el proceso del if-then-elsif-then-else.

CASE [ **TRUE** | [*selector_variable*]]
WHEN [*criterio1* | *expresión1*] THEN
*criterio1_declaraciones*;
WHEN [*criterio2* | *expresión2*] THEN
*criterion2_declaraciones*;
WHEN [*criterio(n+1)* | *expresión(n+1)*] THEN
*criterio(n+1)_declaraciones*;
ELSE
*block_statements*;
END CASE;.

# Estructuras de Control



- Ejercicio

```
BEGIN
CASE TRUE
WHEN (1 > 3) THEN
dbms_output.put_line('Uno es mayor que tres');
WHEN (3 < 5) THEN
dbms_output.put_line('Tres es menor que cinco');
WHEN (1 = 2) THEN
dbms_output.put_line('Uno es igual a dos');
ELSE
dbms_output.put_line('Nada es cierto.');
END CASE;
END;
/
```

# Cursores



- El resultado de una consulta se guarda en una **estructura denominada cursor**.
- Los cursores se utilizan para guardar el resultado de una consulta. Podemos distinguir dos tipos de cursores:

# Cursores



- **Cursores implícitos**. Este tipo de cursores se utiliza para operaciones **SELECT INTO**. Se usan cuando la consulta devuelve un único registro.
- **Cursores explícitos**.  Se utilizan cuando la consulta devuelve un conjunto de registros. Ocasionalmente también se utilizan en consultas que devuelven un único registro por razones de eficiencia, debido a que lo hacen más rápido.

# Cursores



- Sintaxis cursor explícitos.

  **CURSOR *nombre_cursor* IS**

  *instrucción_SELECT*

- Para abrir el cursor

  **OPEN *nombre_cursor*;**

  o bien (en el caso de un cursor con parámetros)

  **OPEN *nombre_cursor*(*valor1, valor2, ..., valorN);***

# Cursores



- Para recuperar los datos en variables

  **FETCH nombre_cursor INTO** lista_variables;

  -- o bien ...

  **FETCH nombre_cursor INTO** registro_PL/SQL(%rowtype);

- Para cerrar el cursor

  **CLOSE nombre_cursor**;

# Procesamiento cursor explícito



**Consta de 4 pasos:**

– **1. Declaración del cursor:** En la sección DECLARE:
**CURSOR <Nombre_Cursor> IS <Orden_SELECT>;**

• La orden_**SELECT puede utilizar variables declaradas antes.**

– **2. Apertura: OPEN <Nombre_Cursor>;**

• Ejecuta la consulta asociada a ese cursor con el valor actual de las variables que use y pone el puntero apuntando a la primera fila.

– **3. Extraer datos: Es similar a la cláusula INTO de la orden SELECT:**
**FETCH <Nombre_Cursor> INTO {<Variables> | <Registro>};**

• Después de cada **FETCH el puntero avanza a la siguiente fila.**

• El **atributo <Nombre_Cursor>%FOUND vale TRUE si no se ha** llegado al final del cursor. El opuesto es **%NOTFOUND.**

– **4. Cierre: CLOSE <Nombre_Cursor>;**

• Libera la memoria de un cursor abierto. Tras esto no podrá usarse.

# Atributos cursores



- **<Nombre_Cursor>%FOUND: Atributo booleano que devuelve:**
  - **TRUE si la última orden FETCH devolvió una fila.**
  - **FALSE en caso contrario.**
  - **NULL si aún no se a efectuado ninguna extracción con FETCH.**
  - Genera un **ERROR si el cursor no está abierto (ORA-1001, INVALID_CURSOR).**
- • **<Nombre_Cursor>%NOTFOUND: Atributo booleano opuesto a %FOUND (en los valores lógicos).**
  - Suele usarse como condición de salida en un bucle.
  - Cuando toma el valor **TRUE la anterior orden FETCH no se realizó**
  - -- asignación a las variables de **INTO, por lo que contendrán lo mismo** que tenían antes (valores de la última fila recuperada).
- **<Nombre_Cursor>%ISOPEN: Atributo booleano que devuelve TRUE** si el cursor está abierto. En caso contrario devuelve **FALSE.**
  - Nunca genera un error.
- **<Nombre_Cursor>%ROWCOUNT: Atributo numérico que devuelve el** número de filas extraídas con **FETCH hasta el momento.**
  - Genera un **ERROR si el cursor no está abierto.**

# Cursores



- Ejemplo cursor implicito

```
  DECLARE
vdescripcion VARCHAR2(50);
BEGIN
    SELECT DESCRIPCION
   INTO vdescripcion
from PAISES
WHERE CO_PAIS = 'ESP';
dbms_output.put_line('La lectura del cursor es: '
|| vdescripcion);
end;
 /
```

# Cursores



- Ejercicio

Adecue el código anterior a las tabla(s) con las ha venido trabajando.

# Cursores



- Ejemplo cursor explicito (1/2)

```
set serveroutput on;
DECLARE
 CURSOR c_paises
 is
 select co_pais,descripcion,continente
 from paises;
 v_pais varchar2(3);
 v_descripcion varchar2(50);
 v_continente varchar2(25);
 BEGIN
```

# Cursores



- Ejemplo cursor explicito (2/2)

```
OPEN c_paises;
 FETCH c_paises INTO v_pais,v_descripcion,v_continente;
 while c_paises%FOUND
 LOOP
 dbms_output.put_line(chr(13));
 dbms_output.put_line('la lectura del cursos es:'||v_pais||'
 '||v_descripcion||' '||v_continente);
 FETCH c_paises INTO v_pais,v_descripcion,v_continente;
 END LOOP;
CLOSE c_paises;
END;
```

# Cursores



- Ejercicio

Adecue el código anterior a las tabla(s) con las ha venido trabajando.

# Cursores



## Ejemplo cursor explicito (1/2)

```
set serveroutput on;
DECLARE
 CURSOR c_paises
 is
 select co_pais,descripcion,continente
 from paises;
 vr_paises c_paises%ROWTYPE;
 BEGIN
     OPEN c_paises;
```

# Cursores



## Ejemplo cursor explicito (2/2)

```
      LOOP
        FETCH c_paises INTO vr_paises;
    exit when c_paises%NOTFOUND;
        dbms_output.put_line('la lectura del cursos es:'||
vr_paises.co_pais||' '||vr_paises.descripcion||' '||
vr_paises.continente);
    END LOOP;
   CLOSE c_paises;
  END;
```

# Cursores



- Ejercicio

Adecue el código anterior a las tabla(s) con las ha venido trabajando.

# EJERCICIOS



```
set serveroutput on;
BEGIN
2 DBMS_OUTPUT.PUT_LINE (‘HOLA MUNDO’);
3 END;
4 /
```

1. Inicialmente ejecutelo en la línea de comandos.
2. Después guardelo en un archivo y ejecute el script

# EJERCICIOS



```
DECLARE
d DATE := SYSTIMESTAMP;
t TIMESTAMP(3) := SYSTIMESTAMP;
BEGIN
  dbms_output.put_line('DATE ['||d||']');
  dbms_output.put_line('TO_CHAR ['||
TO_CHAR(d,'DD-MON-YY HH24:MI:SS')||']');
   dbms_output.put_line('TIMESTAMP ['||t||']');
 END;
/
-- to_char=convierte un número o fecha a cadena
```

# EJERCICIOS



```
DECLARE
nombre VARCHAR2(30);
BEGIN
 nombre := '&entrada';
dbms_output.put_line('Hola '|| nombre );
END;
/
```

# EJERCICIOS



```
DECLARE
CURSOR c_ejercicio IS
SELECT nombrearticulo FROM articulo;
BEGIN
FOR i IN c_ejercicio LOOP
dbms_output.put_line('El titulo es ['||
i.nombrearticulo||']');
END LOOP;
END;
/** Ajuste este código a la estructura de la base de datos
con la que se ha venido trabajando
```

# EJERCICIOS



```
DECLARE
var_numero NUMBER := &numero;
var_resultado NUMBER;
BEGIN
var_resultado := POWER(var_numero, 2);
DBMS_OUTPUT.PUT_LINE (‘El resultado es: '||
var_resultado);
END;
```

a)Si el valor de var_numero es igual a 10, ¿Cuál es la salida en la pantalla?

b) ¿Cuál considera que sea el propósito de utilizar una substitución de variable (signo de &)?

# EJERCICIO



1. Cree una tabla llamada trabajadores con tres campos, nombre, salarioxhora y horas de trabajo
2. Inserte  10 registros dentro de esta tabla
3. Cree un cursor explicito que se llame ctrabajador.,  que consulte y muestre en pantalla los datos almacenados en la tabla.

# Ejercicios



```
SET SERVEROUTPUT ON
DECLARE
  CURSOR ctrabajador
  IS
  SELECT nombre,  SALPORHORA, HORASTRABAJADAS
  FROM trabajadores;
  vnombre VARCHAR2(50);
  vsalporhora number(8,2);
  vhorastrabajadas  number(10);
BEGIN
  OPEN ctrabajador;
LOOP
  FETCH ctrabajador INTO vnombre,vsalporhora,vhorastrabajadas;
  EXIT WHEN ctrabajador%NOTFOUND;
  dbms_output.put_line('El registro es: ['||vnombre||' '||vsalporhora||' '||vhorastrabajadas ||']');
  END LOOP;
  CLOSE ctrabajador;
END;
/
```

# Ejercicios



•Del ejercicio anterior, intercambie las variables por un registro. Los registros cuales son agrupaciones de datos relacionados similares a las estructuras (**struct) del lenguaje C o los registros de Modula2.**

Al Igual que en los lenguajes de programación, para acceder a un campo se usa la **Notación Punto:**

**<VariableRegistro>.<Campoi>.**

– Es posible **asignar Registros: si son del mismo tipo.**

• Si son de tipos distintos no se pueden asignar, aunque estén definidos igual. En ese caso, se pueden asignar campo a campo.

• También se pueden asignar los campos de un **SELECT en un registro compatible.**

– Declarar registros con los campos de una tabla**: <Tabla> %ROWTYPE**

# Ejercicios



- Retome el cursor explícito ctrabajador y ahora defínalo con un parámetro

```
CURSOR ctrabajador(p_nombre VARCHAR2)
        condicion = p_nombre
        OPEN ctrabajador(‘ valor’)
```

- Agregue substitución de variable en el cursor explícito con parámetro.

# Subprogramas



**Son bloques PL/SQL con nombre que:**

– Se almacenan en la BD: Los bloques anónimos (o nominados) no se almacenan.

– Se compilan una vez cuando se definen: Los bloques anónimos (o nominados) se compilan cada vez que se ejecutan.

– Pueden ejecutarse o llamarse desde otros bloques PL/SQL.

– Dos Tipos:

**Procedimientos (PROCEDURE) o**
**Funciones (FUNCTION).**

– **Borrar un Subprograma:**

**DROP {PROCEDURE|FUNCTION} <Nombre_S>;**

# Procedimientos



- Tienen la siguiente estructura general:

**Procedure** <nombre procedimiento>
[(<lista de parámetros>)]
**IS**
<declaraciones>;
**BEGIN**
<instrucciones>;
**EXCEPTION**
<excepciones>;
**END** [<nombre procedimiento>];

# Procedimientos



En esta estructura se aprecian dos partes:

- **La cabecera o especificación del procedimiento**, que comienza con la palabra PROCEDURE y termina despúes del último parámetro
- **El cuerpo del procedimiento**. Corresponde con un bloque PL/SQL. Comienza con la palabra IS (o AS) y termina con la palabra END, opcionalmente seguida del nombre del procedimiento.
- En la lista de parámetros se encuentra la declaración de cada uno de los parámetros separados por comas. El formato genérico es:

**<nombre variable> [IN | OUT | IN OUT]<tipo dato>[{:=Default} <valor>]**

# Procedimientos



Para crear o reemplazar un procedimiento se ocupa el siguiente comando:

```
CREATE [OR REPLACE] PROCEDURE <NombreP> [(
   <Argumento1> [IN|OUT|IN OUT] <Tipo1>,
   ...
   <ArgumentoN> [IN|OUT|IN OUT] <TipoN>)] {IS|AS}
<Bloque_del_Procedimiento_SIN_la_palabra_DECLARE>;
```

Los **Argumentos son opcionales y pueden declararse de 3 modos:**

- **IN** (opción por defecto): Argumento de Entrada. Es de *sólo lectura.*

*(paso por valor)*

- **OUT:** Argumento de Salida. Es de *sólo escritura (sólo puede asignarsele valores). (paso por referencia)*

# Procedimientos



Es un paso de parámetros por variable (al terminar el procedimiento se copia el valor de ese argumento en el argumento actual que aparece en la llamada).

• **IN OUT:** Argumento de Entrada y Salida. Es un argumento tipo OUT, pero que puede también leerse. El argumento actual también debe ser una variable. (paso por referencia)

– La Llamada se efectúa como en un lenguaje de programación normal

# Ejemplo procedimiento IN



```
create or replace procedure insdato
(v_nombre IN trabajadores.nombre%TYPE, v_salxhora IN
trabajadores.salporhora%TYPE, v_horatrab IN
trabajadores.horastrabajadas%TYPE)
IS
BEGIN
INSERT INTO trabajadores (nombre,salporhora,horastrabajadas)
values (v_nombre,v_salxhora,v_horatrab);
commit work;
end insdato;
/
SQL>  @ /home/coria/sqlcodigo/procedimientoin.pls
Procedure created.
SQL> exec insdato('javier garduño lopez', 30,50)
PL/SQL procedure successfully completed.
```

# Ejemplo procedimiento (OUT)



```
create or replace procedure consdato
(v_nombre    IN VARCHAR2,
 v_salxhora OUT NUMBER,
 v_horatrab OUT NUMBER)
AS
BEGIN
select salporhora,horastrabajadas
into v_salxhora,v_horatrab
from trabajadores
where nombre=v_nombre;
END consdato;
/
SQL> @ /home/coria/sqlcodigo/procedimientout.pls
Procedure created.
```

# Ejemplo procedimiento (OUT)



```
SET SERVEROUTPUT ON
DECLARE
v_salxhora  trabajadores. salxhora%TYPE;
v_horatrab  trabajadores.horastrabajadas%TYPE;
BEGIN
consdato('javier garduno lopez',v_salxhora,v_horatrab);
dbms_output.put_line('El empleado javier garduño lopez tiene un
salario x hora de:' ||'
 '||  v_salxhora ||' '|| 'y trabaja ' ||' '|| v_horatrab ||' '|| 'horas');
END;
/
SQL>  @ /home/coria/sqlcodigo/llamaprocedimientout.pls
El empleado javier garduno lopez tiene un salario x hora de: 30 y
trabaja  50 horas
PL/SQL procedure successfully completed.
```

# Ejercicio procedimiento (IN/OUT)



1. Cree las tabla estudiantes con tres campos, numero de cuenta, nombre, apellidos – considere el número de cuenta como llave primaria -.
2. Inserte 10 registros a esta tabla.
3. Cree un procedimiento pbuscaalumno, que tome como entrada el número de cuenta y de salida el nombre y apellidos.
4. Cree un bloque anónimo, desde el cual mande llamar el procedimiento pbuscaalumno y lo imprima en pantalla.

5. Explique la relación entre los parámetros que se encuentran en el la definición del encabezado del procedimiento con respecto a los parámetros que son pasados dentro y fuera del procedimiento

# PRACTICA 1



1. Escriba dos script's (uno para crear la estructura de las tablas –incluyendo restricciones de integridad– y otro para insertar los datos) que permitan introducir la siguiente información a la base de datos:

| cat_divisiones | |
|---|---|
| id_division N(2) | descripción_division V(20) |
| 1 | NORTE |
| 2 | SUR |
| 3 | ESTE |
| 4 | OESTE |

| cat_clientes | | | |
|---|---|---|---|
| id_cliente N(2) | razon_social_cliente V(50) | id_division N(2) | dias_credito_cliente N(2) |
| 1 | CLIENTE1 | 1 | 15 |
| 2 | CLIENTE2 | 1 | 30 |
| 3 | CLIENTE3 | 2 | 15 |

| facturas | | | | | |
|---|---|---|---|---|---|
| id_factura N(3) | fecha_factura D | id_cliente N(2) | monto_factura N(10,2) | iva_factura N(10,2) | total_factura N(10,2) |
| 1 | 04/09/2006 | 2 | 300 | 45 | 345 |
| 2 | 13/09/2006 | 1 | 1000 | 150 | 1150 |
| 3 | 20/09/2006 | 1 | 3000 | 450 | 3450 |
| 4 | 25/09/2006 | 3 | 600 | 90 | 690 |

| pagos | | | |
|---|---|---|---|
| id_pago N(3) | fecha_pago D | id_cliente N(2) | importe_pago N(10,2) |
| 1 | 01/10/2006 | 3 | 60 |

# PRACTICA 1



2. Escriba una consulta que muestre todos los datos del cliente, la descripción de la división a la que está asignado y las facturas que tiene registradas (se deben visualizar todos los clientes, aún cuando no tengan facturas).
3. Escriba una consulta similar a la de la pregunta anterior, pero que en lugar de las facturas, muestre los pagos.
4. Cree una vista a partir de la unión de las consultas de los puntos 2 y 3 llamada vista_clientes_facturas_pagos, en la cual se diferencien perfectamente las facturas de los pagos.
5. Cree una consulta a partir de la vista creada en el punto anterior que proporcione los totales de facturación, de pagos y el saldo de los clientes.
6. Cree una consulta que muestre los datos de las facturas vencidas, incluyendo la razón social del cliente al que pertenecen y los días de atraso en el pago.

# PRACTICA 1



1. Construya un algoritmo en PL/SQL que divida el texto "**ESTE ES UN CURSO DE NIVEL INTERMEDIO DE PL/SQL QUE SIRVE PARA INTRODUCIR AL PARTICIPANTE EN LA TECNOLOGIA ORACLE**" en dos cadenas, una con longitud lo más cercana posible a **60** caracteres (**sin excederse**) y la otra con el resto de la cadena (la división debe hacerse justamente en un espacio en blanco, y debe servir para cualquier cadena de longitud **120**). **Almacene** las dos cadenas resultantes en una tabla temporal creada por usted.

2. Utilice la tabla **pagos** creada en la **Práctica 1** para construir un algoritmo en PL/SQL que eleve a la tercera potencia el **importe_pago** del pago con **identificador 1**; el resultado de tal operación debe **guardarse** en la tabla como **incremento** de dicho importe.

3. Utilice la tabla **facturas** creada en la **Práctica 1** para construir un algoritmo en PL/SQL que divida en unidades, decenas, centenas, etc., el **total_factura** de la fila con **identificador igual a 3**; **almacene** sus resultados en la tabla temporal creada en el problema 1.

# PRACTICA 1



De la tabla trabajadores construya la siguiente salida, tomando como base las horas trabajadas y el sueldo por hora: (sin utilizar varray)
El empleado : [ oscar caballero martinez]
tiene un sueldo sin impuestos de: [ $2,432.00]
Sueldo mas iva: [ $2,796.00]
Sueldo menos 10% isr, y 10 % iva: [ $2,309.00]
su sueldo neto es de: [dos mil trescientos diez]
==========
El empleado : [ jose antonio coria fernandez]
tiene un sueldo sin impuestos de: [ $2,686.00]
Sueldo mas iva: [ $3,088.00]
Sueldo menos 10% isr, y 10 % iva: [ $2,550.00]
su sueldo neto es de: [dos mil quinientos cincuenta y uno]
==========

# EJERCICIOS



Evalúe las siguientes declaraciones y determine cual de estas no es válida y porque.

a)

```
Declare
 v_id  Number(4);
```

b)

```
 Declare
v_x,v_y,v_z VARCHAR2(10);
```

c)

```
  Declare
     v_cumpleanio DATE NOT NULL;
```

d)

```
  Declare
     v_boolean:= 2;
```

# EJERCICIOS



Del siguiente bloque anónimo, encuentre los errores y efectúe las correcciones requeridas.

```
DECLARE
    v_longitud_nr NUMBER :=5.5;
    v_ancho_nr  NUMBER :=3.5;
    v_area_nr   NUMBER;
BEGIN
    v_area_nr:=v_longitud_nr*v_ancho_nr
    DBMS_OUTPUT.put_line('Area:'||area_nr);
END;
/
```

# EJERCICIOS



Capture y ejecute el siguiente código,

```
DECLARE
       v_peso NUMBER(3) := 600;
       v_mensaje  varchar2(255):='Producto 10012'
BEGIN
     DECLARE
        v_peso NUMBER(3):=1;
        v_mensaje:= VARCHAR2(255):='Producto 11001';
        v_nuevo_art varchar2(50):='America';
      BEGIN
        v_peso=v_peso+1;
        v_nuevo_art:='Europeo' || v_nuevo_art;
      END;
v_peso:=v_peso+1;
v_mensaje:=v_mensaje || 'esta en almacen';
v_nuevo_art:='Europeo'||v_nuevo_art;
END;
```

# EJERCICIOS



Del código de la página anterior, determine lo siguiente:

1. El valor de v_peso dentro del sub-bloque
2. El valor de v_nuevo_art dentro del sub-bloque
3. El valor de v_peso dentro del bloque principal
4. El valor de v_nuevo_art dentro del bloque principal.

Advierta los bloques anidados y alcance de las variables. Verificando lo siguiente:

- Una declaración puede ser anidada en cualquier parte donde una declaración de instrucciones es permitida.
- Un bloque anidado se convierte en una declaración.
- El alcance de un objeto es la región de un programa el cual puede referir al objeto
- El identificador es visible en el bloque en cual es declarado y todos los subbloques anidados. Si el bloque no encuentra el identificador declarado localmente, busca en la sección declarativa del bloque padre. El bloque no busca en los bloques hijos.

# EJERCICIOS



Capture y ejecute el siguiente código. Realize los cambios necesarios para que despliegue en orden de mayor a menor, teniendo como valores iniciales v_num1=2 y v_num2=10.

```
DECLARE
  v_num1 NUMBER := 5;
  v_num2 NUMBER := 3;
  v_temp NUMBER;
BEGIN
  IF v_num1 > v_num2 THEN
    v_temp := v_num1;
    v_num1 := v_num2;
    v_num2 := v_temp;
  END IF;
  DBMS_OUTPUT.PUT_LINE ('Valor1 = '||v_num1);
  DBMS_OUTPUT.PUT_LINE ('Valor2 = '||v_num2);
END;
```

# EJERCICIOS



Capture y ejecute el siguiente código. Modifique el código con IF-THEN-ELSE. Si la fecha Especificada no coincide con que sea fin de semana, despliegue un mensaje que lo notifique.

```
SET SERVEROUTPUT ON
DECLARE
  v_fecha DATE := TO_DATE('&sv_usuario_dia', 'DD-MON-YYYY');
  v_dia  VARCHAR2(15);
BEGIN
    v_dia := RTRIM(TO_CHAR(v_fecha, 'DAY'));
    IF v_dia IN ('SATURDAY', 'SUNDAY') THEN
    DBMS_OUTPUT.PUT_LINE (v_dia||' es fin de semana');
  END IF;
    DBMS_OUTPUT.PUT_LINE ('Termino');
END;
```

# EJERCICIOS



Capture y ejecute el siguiente código. Posteriormente, cambie la línea 8 después de la línea 12, ¿cuantos valores son desplegados?, sustituya las líneas 10-12 por la siguiente línea EXIT WHEN v_counter = 5, ¿que efecto tiene? y finalmente elimine las líneas 10 y 12 ¿que ocurre?

```
1 SET SERVEROUTPUT ON
2 DECLARE
3 v_contador BINARY_INTEGER := 0;
4 BEGIN
5 LOOP
6   -- incrementa el contador en uno
7   v_contador := v_contador + 1;
8   DBMS_OUTPUT.PUT_LINE ('v_contador = '||v_contador);
9   -- si la condición se cumple termina el ciclo
10   IF v_contador = 5 THEN
11    EXIT;
12   END IF;
13 END LOOP;
14 -- Termina
15 DBMS_OUTPUT.PUT_LINE ('Finalizado…');
16 END;
```

# EJERCICIOS



Capture y ejecute el siguientes código. Modifique el código eliminando las líneas 3 y 5, de la línea 6 solo deje el CASE, y la linea 7 sustituya el 0 por mod(v_numero,2)=0, ¿que ocurre? .

```
1DECLARE
2 v_numero NUMBER := &sv_numero;
3 v_numero_flag NUMBER;
4 BEGIN
5 v_numero_flag:=mod(v_numero,2);
6 CASE v_numero_flag
7 WHEN  0 THEN
8 DBMS_OUTPUT.PUT_LINE (v_numero||' es un numero par');
9 ELSE
10 DBMS_OUTPUT.PUT_LINE (v_numero||' es un numero impar');
11 END CASE;
12 DBMS_OUTPUT.PUT_LINE ('Finalizo');
13 END;
14 /
```

# EJERCICIOS



Capture y ejecute el siguientes código. Modifique el código eliminando las líneas 3 y 5, de la línea 6 solo deje el CASE, y la linea 7 sustituya el 0 por mod(v_numero,2)=0, ¿que ocurre? .

```
1DECLARE
2 v_numero NUMBER := &sv_numero;
3 v_numero_flag NUMBER;
4 BEGIN
5 v_numero_flag:=mod(v_numero,2);
6 CASE v_numero_flag
7 WHEN  0 THEN
8 DBMS_OUTPUT.PUT_LINE (v_numero||' es un numero par');
9 ELSE
10 DBMS_OUTPUT.PUT_LINE (v_numero||' es un numero impar');
11 END CASE;
12 DBMS_OUTPUT.PUT_LINE ('Finalizo');
13 END;
14 /
```

# EJERCICIOS



Capture y ejecute el siguientes código. Corrija el error que presenta este programa.

¿Cuál es el valor predeterminado de v_numero_flag?.

```
DECLARE
v_numero NUMBER := &sv_numero;
v_numero_flag Boolean;
BEGIN
CASE v_numero_flag
WHEN MOD(v_num,2) = 0 THEN
DBMS_OUTPUT.PUT_LINE (v_num||' is even number');
ELSE
DBMS_OUTPUT.PUT_LINE (v_num||' is odd number');
END CASE;
DBMS_OUTPUT.PUT_LINE ('Done');
END;
```

# EJERCICIOS



- Utilice la declaración CASE para desplegar el nombre del día en la pantalla basándose en el número de día de la semana. Es decir, si el numero del día de semana es 5, se desplegaría Jueves.

# EJERCICIOS



Capture y ejecute el siguiente código. Utilizando WHILE construya un programa que sume los número enteros entre el 1 y el 20.

declare

```
DECLARE
  v_contador NUMBER := 1;
BEGIN
  WHILE v_contador < 5 LOOP
    DBMS_OUTPUT.PUT_LINE('contador = '||v_contador);
      v_contador := v_contador - 1;
  END LOOP;
END;
```

# EJERCICIOS



Capture y ejecute el siguiente código. Posteriormente, cree un programa que calcule el factorial de 10, emplando FOR...LOOP

```
SET SERVEROUTPUT ON
DECLARE
v_suma NUMBER:=0;
BEGIN
FOR v_contador IN REVERSE 1..5 LOOP
v_suma:=v_suma+v_contador;
DBMS_OUTPUT.PUT_LINE ('v_contador = '||v_contador);
END LOOP;
DBMS_OUTPUT.PUT_LINE ('La suma es = '||v_suma);
END;
~
```

# EJERCICIOS



Capture y ejecute el siguiente código. Modifique el programa de tal forma que nos devuelve el total de segundos.

```
declare
v_hora_nr NUMBER:=12;
v_minuto_nr NUMBER:=20;
procedure p_minutos (i_fecha_dt DATE,o_hora_nr IN OUT NUMBER, o_minuto_nr IN OUT
NUMBER)
is
begin
DBMS_OUTPUT.put_line(o_hora_nr||'/'||o_minuto_nr);
-- toma la hora del sistema , se convierte a número
o_hora_nr:=to_NUMBER(to_char(i_fecha_dt,'hh24'));
-- se toman los minutos de la hora del sistema
o_minuto_nr :=to_char(i_fecha_dt,'mi');
DBMS_OUTPUT.put_line(o_hora_nr||'/'||o_minuto_nr);
end;
begin
p_minutos(sysdate, v_hora_nr, v_minuto_nr);
DBMS_OUTPUT.put_line ('Total minutos:'||(v_hora_nr*60+v_minuto_nr));
end;
/
```

# EJERCICIOS



Capture, ejecute y analice el siguiente código. Incluya la condición que evalue si se pasa un valor nulo y que finalice el procedimiento si la condición se cumple.

```
declare
   v_texto VARCHAR2(50):= 'Imprime la  <in> linea';
   procedure p_imprime
        (i_cadena_texto  in VARCHAR2,
         i_reemplazo_texto in VARCHAR2 := 'nueva')
   is
   begin
      DBMS_OUTPUT.put_line(replace(i_cadena_texto,
                          '<in>', i_reemplazo_texto));
   end p_imprime;
begin
   p_imprime (v_texto,'primera');
   p_imprime (v_texto,'segunda');
   p_imprime (v_texto);
end;
```

# EJERCICIOS



Capture y ejecute el siguiente código. Modifique el programa de tal forma que devuelva la longitud de la cadena “Este diciembre sera muy frio”. En la linea 5 agregue CREATE OR REPLACE antes de PROCEDURE, ¿Que ocurre?, ¿Cual sería la explicación de lo sucedido?

```
1 DECLARE
2in_cadena VARCHAR2(100) := 'Cadena de prueba';
3 out_cadena VARCHAR2(200);
4 --declaración del procedimiento y definición---
5   PROCEDURE doble (original IN VARCHAR2, nueva_cadena OUT VARCHAR2)
6  AS
7    BEGIN
8    nueva_cadena := original || original;
9 END doble;
10  BEGIN
11 DBMS_OUTPUT.PUT_LINE ('in_cadena: ' ||' '|| in_cadena);
12       doble (in_cadena, out_cadena);
13 DBMS_OUTPUT.PUT_LINE ('out_cadena: ' ||' '|| out_cadena);
14 END;
15 /
```