CONSTRUCCIÓN DE SCRIPTS CON PL/SQL.

GBDD

¿POR QUÉ PL/SQL?

- Hasta ahora se ha visto cómo interrogar a la base de datos utilizando las sentencias SQL.
- SQL ofrece una gran potencia para interrogar y administrar la base de datos.
- Sin embargo, se puede percibir que hay ciertos tipos de preguntas o acciones que no es posible realizar, se necesita un lenguaje más potente, para ello **ORACLE** desarrolla un lenguaje procedimental, que va a extender la potencia que ofrece SQL, y el resultado es el **PL/SQL** (Procedural Language/Structured Query Language).
- Un lenguaje de programación **incrustado** en **Oracle**.
- **PL/SQL** soporta el lenguaje de consultas, es decir el SQL, pero **no soporta** ordenes de definición de datos (**DDL**) ni de control de datos (**DCL**).

RECORDAMOS:

- **Lenguaje de Definición de Datos(DDL):**
  - CREATE.
  - ALTER.
  - DROP.
- **Lenguaje de Manipulación de Datos (DML):**
  - SELECT.
  - INSERT.
  - UPDATE.
  - DELETE.
- **Lenguaje de Control de Datos (DCL):**
  - GRANT.
  - REVOKE.

ORACLE
PL SQL

¿POR QUÉ PL/SQL?

- PL/SQL además va a incluir las características propias de un lenguaje procedimental:
  - 1. El uso de variables.
  - 2. Estructuras de control de flujo y toma de decisiones.
  - 3. Control de excepciones.
  - 4. Reutilización del código a través de paquetes, procedimientos y funciones.

ORACLE
PL SQL

¿POR QUÉ PL/SQL?

- Podremos escribir **procedimientos** o **funciones**, o bien pueden escribir bloques de código anónimo

A tener en cuenta: Los bloques anónimos no se quedan almacenados en el servidor.

- El código **PL/SQL** es ejecutado en el **servidor**, lo que supone un **ahorro de recursos** a los clientes.
- Otro código especial es el uso de **disparadores**, también conocidos como **triggers**, que permiten realizar una acción concreta sobre la base de datos cuando se ha **modificado**, **insertado** o **borrado** un registro de una tabla.

ORACLE
PL SQL

EJEMPLO DE DISPARADOR

```
CREATE OR REPLACE TRIGGER MI_DISPARADOR IS
    BEFORE INSERT OR UPDATE OF numero ON tabla_temporal
    FOR EACH ROW
BEGIN
    IF :new.numero < 0 THEN
          RAISE_APPLICATION_ERROR(-20100,'¡¡¡¡Error!!!');
END;
```

La **estructura** de un **BLOQUE ANÓNIMO** en PL/SQL es la siguiente:

```
[DECLARE]
-- variables, cursores, excepciones definidas por el usuario
BEGIN
        --sentencias SQL
        -- sentencias de control PL/SQL;
[EXCEPTION]
        -- acciones a realizar cuando se producen errores END;
/ -- se utiliza para ejecutar el código.
```

Recuerda:

- De acuerdo con la sintaxis, lo único **obligatorio** para crear un **BLOQUE ANÓNIMO** en PL/SQL son las palabra clave **BEGIN** Y **END;**.
- En el apartado **DECLARE** se van a declarar todas las **variables, constantes, cursores** y **excepciones** definidas por el usuario.
- **Todas las sentencias se escribirán una detrás de otra separadas por punto y coma ';'**
- A lo largo de este capítulo se irá explicando para qué sirve cada uno de estos elementos, que serán de vital importancia para aprovechar la potencia del PL/SQL.

- Ejemplo de **bloque anónimo** para mostrar el mensaje de Hola Mundo

```
SET SERVEROUTPUT ON
BEGIN
    DBMS_OUTPUT.PUT_LINE ('Hola Mundo');
END;
/
```

Salida por pantalla de los resultados de una ejecución

Para poder realizar una visualización en pantalla de la ejecución y resultados internos del código PL/SQL, hay que utilizar la invocación de un paquete incluido en Oracle PL/SQL denominado DBMS_OUTPUT, el cual permite dirigir a pantalla resultados mediante el uso de la función PUT_LINE.

No obstante, para que se haga completamente efectiva dicha salida a pantalla, antes hay que activar el comando de ejecución en pantalla del paquete DBMS_OUTPUT, siempre que se inicie una nueva sesión con la herramienta correspondiente. Este comando es el siguiente:

```
SET SERVEROUTPUT ON;
```

Otra forma de activar la salida

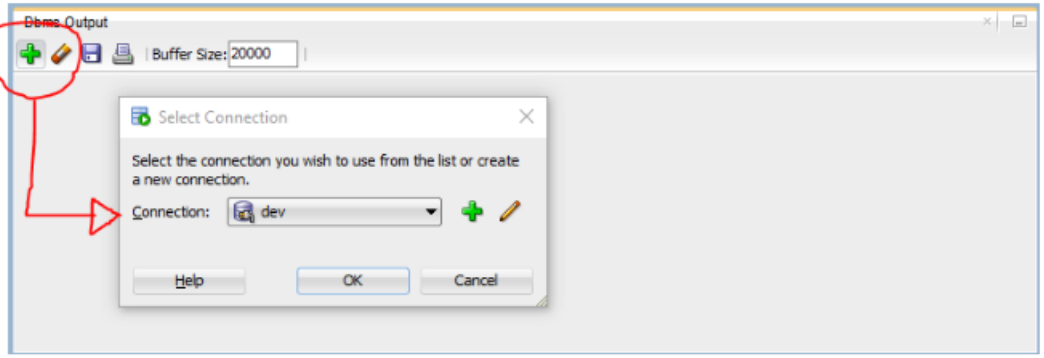
Para poder ver la salida de `DBMS_OUTPUT` en SQL Developer, debes seguir las 2 etapas siguientes:

1. Abrir la ventana **Dbms Output**.

   Esto lo haces desde el menú: **View --> Dbms Output**.

2. Activar `DBMS_OUTPUT` para la conexión a la base de datos.

   Para esto, pulsa el icono +, selecciona la conexión adecuada y pulsa **OK**.



Todo debería funcionar correctamente ahora.

Tipos de datos en PL/SQL

- Los TIPOS de DATOS más comunes son:
  - **NUMBER (Numérico):**
    - Almacena números enteros o de punto flotante, de cualquier longitud.
    - Puede ser especificada la precisión (Número de dígitos) y la escala que es la que determina el número de decimales. NUMBER (5,3) tendría una precisión de 5 dígitos de los cuales 3 serían posiciones decimales.
  - **CHAR (Carácter):**
    - Almacena datos de tipo carácter con una longitud máxima de 32767 y cuyo valor de longitud por defecto es 1.

Tipos de datos en PL/SQL

- Los TIPOS de DATOS más comunes son:
  - **VARCHAR2 (Carácter de longitud variable):**
    - Almacena datos de tipo carácter empleando solo la cantidad necesaria aún cuando la longitud máxima sea mayor.
  - **BOOLEAN (lógico):**
    - Se emplea para almacenar valores TRUE o FALSE.
  - **DATE (Fecha):**
    - Almacena datos de tipo fecha.
    - Las fechas se almacenan internamente como datos numéricos, por lo que es posible realizar operaciones aritméticas con ellas.

- **Declaración de variables**
  - Para poder utilizar variables es necesario antes declararlas en el apartado **DECLARE** del bloque de código anónimo.

Sintaxis:

Nombre_de_variable [CONSTANT] tipo_dato [NOT NULL] [:= | DEFAULT | Expresión];

Declaración de variables

Ejemplos

- Declaramos una variable tipo Fecha y la inicializamos.
  - fechaNacimiento DATE:= '01-SEP-1998';
- Declaramos una variable numérica y la inicializamos.
  - NumDepartamento NUMBER(2) NOT NULL := 10;
- Declaramos una variable carácter y la inicializamos.
  - localidadCliente VARCHAR2(13) := 'Talavera de la Reina';
- Declaramos una constante numérica.
  - fijoComision CONSTANT NUMBER := 500;

Recuerda:

- Las **variables** y **constantes** tienen que ser de un tipo de dato soportado por el lenguaje.
- Las **variables** son datos cuyos valores van a poder cambiar a lo largo de la ejecución del programa.
- Las **constantes** permanecen inalterables.
- Al elegir el **tipo de dato** para la variable o constante va a condicionar el **formato de almacenamiento**.

Recuerda:

- **La salida**
  - Es necesario activar la variable **SERVEROUTPUT** de Oracle (solamente es necesario hacerlo una vez en la sesión).
    - **SET SERVEROUTPUT ON**
  - Una vez activada podemos escribir por pantalla:
    - dbms_output.put_line (‘Cadena de salida’);
  - Se puede mostrar texto, o el contenido de variables y constantes.
  - Para mostrar texto, simplemente se pone entre comillas simples:
    - dbms_output.put_line('Hola alumnos');

Recuerda

- **La salida**
  - Para mostrar el contenido de una variable, basta con poner el nombre de la variable:
    - dbms_output.put_line(nombreCliente);
  - En la mayoría de las ocasiones interesa mostrar:
    - dbms_output.put_line('Nombre del Cliente: ' II nombreCliente);
  - No importa si la variable fuera de tipo numérico:
    - dbms_output.put_line('Total facturado' II Importe);
  - Incluso pueden ser funciones, cuyo resultado se mostrará:
    - dbms_output.put_line('El mayor es: ' II mayor(4,6));

La entrada

- Para leer valores de teclado hay que hacer una **asignación** a una variable y poner el símbolo **&** seguido de una cadena de caracteres que se mostrará al pedir la entrada.
- **Esta cadena debe ir sin espacios en blanco.**
- Ejemplo:
  - Entrada de valores a variables por teclado nombeVariable := &Texto_a_mostrar;

Ejemplo de bloque anónimo con declaración de variable, petición de datos de tipo texto y salida.

```
DECLARE
    nombre VARCHAR2(25);
BEGIN
    nombre:='&Dime_tu_nombre';
    DBMS_OUTPUT.PUT_LINE ('Hola '|| nombre);
END;
/
```

Ejemplo con petición de datos y salida de tipo

```
SET SERVEROUTPUT ON
--Este código nos pide introducir el valor de altura, Base
-- y calcula el área del triángulo.
DECLARE
Altura INT;
Base INT;
BEGIN
Altura:=&Introduce_Altura;
Base:=&Introduce_Base;
DBMS_OUTPUT.PUT_LINE('UN TRIÁNGULO DE BASE: ' || Base || ' Y DE ALTURA: '
|| ALTURA || ' TIENE UN AREA DE: ' || Base*Altura/2 );
END;
/
```

- Otro ejemplo:

```
SET SERVEROUTPUT ON
--Este código hace la suma, resta, multiplicación
-- y división de dos variables enteras
DECLARE
A INT:= 9;
B INT:= 3;
BEGIN
DBMS_OUTPUT.PUT_LINE(A||' + '||B|| ' = ' ||(A+B));
DBMS_OUTPUT.PUT_LINE(A||' - '||B|| ' = ' ||(A-B)'·
DBMS_OUTPUT.PUT_LINE(A||' * '||B|| ' = ' ||(A*B));
DBMS_OUTPUT.PUT_LINE(A||' / '||B|| ' = ' ||(A/B));
END;
/
```

Estructuras de control

- **La selección**
  - **Sentencia IF**
  - Se trata de evaluar una expresión y en función del valor de que esta expresión sea verdadera o falsa se hacen unas acciones u otras.
  - Sintaxis:
    - IF condición
      THEN instrucciones;
      [ELSIF condición
      THEN instrucciones;]
      [ELSE
      instrucciones;]
      END IF;

Estructuras de control

La selección

- **Sentencia IF**

```
SET SERVEROUTPUT ON
--Este código calcula el máximo entre dos números pasados por consola
DECLARE
A INT;
B INT;
BEGIN
A:=&Introduce_Valor_A;
B:=&Introduce_Valor_B;

IF(A>B) THEN
    DBMS_OUTPUT.PUT_LINE('El valor máximo es el de A: ' ||A);
ELSIF(A<B) THEN
    DBMS_OUTPUT.PUT_LINE('El valor máximo es el de B: ' ||B);
ELSE
    DBMS_OUTPUT.PUT_LINE('A y B son iguales');
END IF;


END;
/
```

Estructuras de control

- **La selección**
  - **Sentencia CASE**
    - Evalúa cada condición hasta encontrar alguna que se cumpla.
    - La sintaxis es:
      - CASE [expresión]

```
WHEN {condicion1 | valor1} THEN
        bloque_instrucciones_1;
WHEN {condicion2 | valor2} THEN
        bloque_instrucciones_2;
…
ELSE
        bloque_instrucciones_por_defecto;
END CASE;
```

Estructuras de control

La selección

- **Sentencia CASE**

```
SET SERVEROUTPUT ON
--Este código
DECLARE
nota INT;

BEGIN
nota:=&Introduce_Valor_nota;

CASE

WHEN nota >=0 AND nota < 5 THEN
    DBMS_OUTPUT.PUT_LINE('INSUFICIENTE');
WHEN nota >=5 AND nota < 6 THEN
    DBMS_OUTPUT.PUT_LINE('SUFICIENTE');
WHEN nota >=7 AND nota < 9 THEN
    DBMS_OUTPUT.PUT_LINE('NOTABLE');
WHEN nota >=9 AND nota <= 10 THEN
    DBMS_OUTPUT.PUT_LINE('SOBRESALIENTE');
ELSE
    DBMS_OUTPUT.PUT_LINE('NOTA NO VÁLIDA');
END CASE;

END;
/
```

Estructuras de control

- **La Iteración**
  - Los bucles repiten una sentencia o un grupo de sentencias varias veces.
  - Hay varios tipos de bucles que dependiendo de:
    - Si se sabe o no el número de veces que se van a repetir las acciones.
    - Si por el contrario, conocemos la condición de repetición o de salida del bucle.
    - En PL/SQL existen tres tipos de bucles:
      - Bucle básico. **LOOP**. Acciones repetitivas sin condiciones globales
      - Bucle **FOR**. Acciones repetitivas basándose en un contador. Número conocido de vueltas.
      - Bucle **WHILE**. Basándose en una condición.

Estructuras de control

- **La Iteración**
  - **Bucle básico LOOP**

    Sintaxis:

    ```
    LOOP
        Instrucciones;
    END LOOP;
    ```

  - Este bucle estaría repitiendo infinitamente las instrucciones.
  - Alguna condición que cuando se cumpla provoque la salida del bucle. Se puede hacer de dos formas:
    - Combinarlo con una condicional **IF**, de forma que cuando se cumpla una condición se fuerza que deje de iterar con la orden **EXIT**.
    - O bien usarlo con la opción **WHEN** y cuando se haga cierta la condición salga del bucle.

Estructuras de control

- **La Iteración**
  - **Bucle básico LOOP**
    - **WHEN:**

```
SET SERVEROUTPUT ON
--Este código realiza 100 iteraciones y sale en la 101
DECLARE
N INT:=0;

BEGIN

LOOP
N:=N+1;
EXIT WHEN N >100;
DBMS_OUTPUT.PUT_LINE(N);
END LOOP;

END;
/
```

Estructuras de control

- **La Iteración**
  - **Bucle WHILE**
    - El bucle **WHILE** se va a estar ejecutando mientras que la condición que se evalúa sea cierta, por lo que dentro del cuerpo del bucle debe de haber alguna instrucción que cambie dicha condición, ya que en caso contrario sería un bucle infinito, el programador debe tener presente esto a la hora de utilizar un bucle WHILE.
    - **Sintaxis:**
      - WHILE condición LOOP

        instrucciones;

        END LOOP;

Estructuras de control

- **La Iteración**
  - **Bucle WHILE**

```
SET SERVEROUTPUT ON
--Este código realiza 99 iteraciones y sale en la 100
DECLARE
Contador INT := 0;
BEGIN
WHILE Contador <= 100 LOOP
    DBMS_OUTPUT.PUT_LINE(Contador);
    Contador := Contador + 1;
END LOOP;
END;
/
```

Estructuras de control

- **La Iteración**
  - **Bucle FOR**
    - Cuando se conoce de antemano el número de repeticiones o iteraciones, sin duda el bucle ideal para hacerlo es **FOR**.
    - Sintaxis:
      - FOR Índice IN [REVERSE] valor_inicial .. valor_final LOOP
        instrucciones;
      END LOOP;

Estructuras de control

- **La Iteración**
  - **Bucle FOR**

```
-- Mostrar los 100 primeros números con un for
DECLARE
i INT;
BEGIN
FOR i IN 0 .. 100 LOOP
    DBMS_OUTPUT.PUT_LINE(i);
END LOOP;
END;
/
```

Estructuras de control

- **La Iteración**
  - **Bucle FOR INVERSO**

```
-- Mostrar los 100 primeros números con un for inverso
DECLARE
i INT;
BEGIN
FOR i IN REVERSE 0 .. 100 LOOP
    DBMS_OUTPUT.PUT_LINE(i);
END LOOP;
END;
/
```

Recuerda:

- PL/SQL no está pensado para interactuar directamente con un **usuario final.**
- Por eso, carece de instrucciones especializadas en la **entrada(teclado)** y **salida**(pantalla).