Índice

1. ESTRUCTURA DE BLOQUES

PL/SQL es un lenguaje de **programación estructurado**. Es un lenguaje procedimental que amplia la funcionalidad de SQL añadiendo estructuras habituales en otros lenguajes de programación, entre las que se encuentran:

- ✓ Variables y tipos
- ✓ Estructuras de control
- ✓ Procedimientos y funciones
- ✓ Tipos de objetos y métodos.

La unidad básica en PL/SQL es el *bloque*. Todos los programas PL/SQL están compuestos por bloques, que pueden estar de forma secuencial o anidados. Normalmente cada bloque realiza una unidad lógica de trabajo en el programa, separando así unas tareas de otras. Hay muchos tipos diferentes de bloques:

Anónimos *(Anonymous blocks)*.- se construyen de forma dinámica y se ejecutan una sola vez.

Con nombre *(Named blocks)*.- son bloques con nombre, que al igual que el anterior se construyen, generalmente, de forma dinámica y se ejecutan una sola vez.

Subprogramas .- procedimientos, paquetes o funciones almacenados en la base de datos. No suelen cambiar después de su construcción y se ejecutan múltiples veces mediante la llamada *call*.

Disparadores *(Triggers)*.- son bloques con nombre que también se almacenan en la base de datos. Tampoco suelen cambiar después de su construcción y se ejecutan varias veces. El suceso de disparo es una orden del lenguaje de manipulación de datos (*Data Manipulation Language*, DML) que se ejecuta sobre una tabla de la base de datos. Entre las órdenes DML podemos citar INSERT, UPDATE y DELETE.

Similar a subprogramas

Todos los bloques tienen tres secciones diferenciadas:

1 *DECLARE*
Declaración de variables, constantes y subprogramas (opcional)

2 *BEGIN*
Sección ejecutable, se incluyen las sentencias de SQL y procedimientos
Es la sección principal del bloque y es **obligatoria**

3 *EXCEPTION*
Zona de excepción, se pueden colocar aquí las sentencias en caso de Error (opcional)
END; /

La única obligatoria es la sección ejecutable, tanto la declarativa como la de manejo de errores son opcionales.

También es obligatorio que dentro de la sección ejecutable exista al menos una orden ejecutable.

Las palabras clave DECLARE, BEGIN, EXCEPTION y END delimitan cada una de las secciones.

El punto y coma final también es obligatorio, ya que forma parte de la sintaxis del bloque.

1. Sección Declarativa.- Donde se localizan todas las variables, cursores y tipos usados por el bloque. También se pueden declarar en esta sección las funciones y procedimientos locales. Estos subprogramas estarán disponibles solo para ese bloque.

2. Sección Ejecutable.- Donde se lleva a cabo el trabajo del bloque. En esta sección pueden aparecer tanto órdenes SQL como órdenes procedimentales.

3. Sección Errores.- El código de esta sección no se ejecutará a menos que ocurra un error.

2. UNIDADES LEXICAS

Una unidad léxica es una secuencia de caracteres, donde los caracteres pertenecen a un conjunto de caracteres permitido en el lenguaje PL/SQL. Este conjunto de caracteres incluye:

- Letras mayúsculas y minúsculas: A – Z y a – z
- Los dígitos: 0 – 9
- Espacios en blanco: tabuladores, caracteres de espaciado y retorno de carro
- Símbolos matemáticos: + - * / < > =
- Símbolos de puntuación: ( ) { } [ ] ¿ ¡ ; : . ‘ “ @ # % ~ & _

3. IDENTIFICADORES

Se emplean para dar **nombre a los objetos PL/SQL**, tales como variables, cursores, tipos y subprogramas. Los identificadores constan de una letra, seguida por una secuencial opcional de caracteres, que pueden incluir letras, números, signos de dólar($), caracteres de subrayado y símbolos de almohadilla(#). Los demás caracteres no pueden emplearse. La longitud máxima de un identificador es de 30 caracteres y todos los caracteres son significativos.

BIEN DECLARADO	MAL DECLARADO
X	X + Y
V_ENAME	_ENAME_
CodEm	COD EM
V1	1V
V2_	ESTA ….. …VARIABLE (+30)
ES_UNA_VARIABLE_#	

Hay que tener en cuenta que PL/SQL no diferencia entre mayúsculas y minúsculas, así todos los siguientes identificadores serian equivalentes desde el punto de vista de PL/SQL:

NUM_EMP
Num_emP
num_emp
nUM_Emp

A la hora de declarar variables hemos de tener en cuenta las *palabras reservadas*, estos son palabra que tienen un significado especial para PL/SQL y no podemos utilizarlas como variables porque tendríamos problemas de compilación, estas palabras son tales como BEGIN, END…

4. DELIMITADORES

Son símbolos que tienen un significado especial para PL/SQL, se utilizan para separar unos identificadores de otro

SIMBOLO	DESCIPCIÓN	SIMBOLO	DESCRIPCIÓN
+	Operador de suma	-	Operador de resta
*	Operador de multiplicación	/	Operador de división
=	Operador de igualdad	<	Operador “menor que”
>	Operador “mayor que”	(	Delimitador inicial de expresión
)	Delimitador final de expresión	;	Terminador de orden
%	Indicador de atributo	,	Separador de elementos
.	Selector de componente	@	Delimitador de enlace a B.D.
‘	Delimitador cadena caracteres	“	Delimitador cadena entrecomill.
:	Indicadorvariable asignación	**	Operador de exponenciación
<>	Operador “distinto de”	¡=	Operador “distinto de”
<=	Operador “menor o igual que”	>=	Operador “mayor o igual que”
:=	Operador de asignación	=>	Operador de asociación
||	Operador de concatenación	--	Comentario, una sola línea
<<	Comienzo de etiqueta	>>	Fin de etiqueta
*/	Cierre de comentario multilínea	<space>	Espacio
<tab>	Carácter de tabulación	<cr>	Retorno de carro

5. LITERALES

a) *Carácter*.- O también denominados *de cadena* constan de uno o mas caracteres delimitados por comillas simples. Se pueden asignar a variables de tipo CHAR o VARCHAR2, sin tener que hacer ningún tipo de conversión:

'12345'
'100%'
…

b) *Numérico*.- Representa un valor entero o real, puede asignarse a una variable de tipo NUMBER sin tener que efectuar conversión alguna. Los literales enteros consisten de una serie de dígitos, precedidos opcionalmente por un signo (+ o -). No se permite utilizar un punto decimal en un literal entero.

123
+7
-9
…

Un literal real consta de signo, opcional, y una serie de dígitos que contiene punto decimal. También pueden escribirse utilizando notación científica.

-17.7
23.0
1.345E7
-7.12e+12

c) **Booleanos**.- Los literales boolenaos representan la verdad o falsedad de una condición y se utilizan en las órdenes IF y LOOP, solo existen tres posibles literales booleanos:

TRUE	Verdadero
FALSE	Falso
NULL	Nulo

6. COMENTARIOS

Monolínea.- Comienza con dos guiones y continúa hasta el final de la línea **--**
Multilínea.- Comienzan con el delimitador ***/**** y terminan con el delimitador ****/***

```
-- comentario de línea

/*comentario
mas comentario */
```

7. VARIABLES

Se definen en el declare, su estructura es:

<*Nombre_identificador*> [constant] **tipo** [not null] [**:=** valor];

Es preferible inicializar una variable siempre que su valor se pueda determinar, no obstante, PL/SQL define el contenido de las variables no inicializadas, asignándolas el valor NULL (valor desconocido o no definido).

Las variables declaradas como Not Null siempre deben ir inicializadas.
La inicialización puede hacerse utilizando :=0 o la palabra reservada DEFAULT.
Las constantes deben ser inicializadas.

```
DECLARE
    Km_a_milla  CONSTANT NUMBER :=1.4;
```

Tipos:

NUMÉRICOS.- Contienen un valor numérico entero o de punto flotante

-Number : numérico
-Dec, decimal: numérico decimal
-Double precision : doble precisión
-Integer, int : enteros
-Real : reales
-Smallint : entero corto
-**Binary_integer**: enteros (importante para los índices en las matrices)
-Natural: numeros naturales
-Positive: números positivos

```
DECLARE
    Dep_num  NUMBER(2)  NOT NULL :=10;
```

CARACTER.- Permite almacenar cadenas o datos de tipo carácter

- **VARCHAR2**.- Pueden contener cadenas de caracteres de longitud variable, su sintaxis es:

varchar2(L)

Donde L es la longitud máxima de la variable, esta longitud se especifica en bytes, no en caracteres. El subtipo VARCHAR es equivalente a VARCHAR2

```
DECLARE
    Ciudad VARCHAR2(10)  :='Ciudad Real';
```

- **CHAR.**- Cadenas de caracteres de longitud fija. La sintaxis es:

  char(L)

  Donde L es la longitud máxima en bytes, pero su especificación no es opcional

- **LONG**.- Es una cadena de longitud variable con una longitud máxima de 32.760 bytes, son muy similares a las variables VARCHAR2

RAW.- Se emplean para almacenar datos binarios

- RAW .- Son similares a las variables CHAR, la sintaxis:

  RAW(L)

  Donde L es la longitud en bytes de la variable, se emplea para almacenar datos binarios de longitud fija.

- LONG RAW .- Son similares a los datos LONG, la longitud máxima de una variable LONG RAW es de 32.760 bytes

ROWID.- En él se puede almacenar un identificador de columna que es una clave que identifica unívocamente a cada fila de la base de datos

BOOLEANOS.- Solo pueden contener los valores TRUE, FALSE o NULL

TIPOS COMPUESTOS.- Hay disponibles dos tipos compuestos: registros y tablas. Un tipo compuesto es aque que consta de una serie de componentes. Una variable de tipo compuesto contendrá una o más variables escalares.

8. UTILIZACIÓN %TYPE

Hay ocasiones que las variables que usamos en PL/SQL se emplean para manipular datos almacenados en una tabla de la base de datos. En estos casos tendrá que ser la variable del mismo tipo que las columnas de las tablas.

El atributo %TYPE se utiliza para declarar una variable con el mismo tipo que una columna de una tabla o que otra variable definida anteriormente.

```
<Nombre_identificador> <tabla.columna>%TYPE;
```

```
var_nombre   empleados.nombre%TYPE;

balance  NUMBER;
balance_minimo balance%TYPE:=10;
```

9. SUBTIPOS DEFINIDOS POR EL USUARIO

Un subtipo es un tipo PL/SQL que se basa en otro tipo existente. Los subtipos pueden emplearse para dar nombres alternativos, no es más que dar un nuevo nombre al tipo de varibale. La sintaxis es:

```
SUBTYPE nuevo_tipo IS tipo_original;
```

10. CONVERSIÓN ENTRE TIPOS DE DATOS

El propio intérprete realiza la conversión de los datos, entre los distintos tipos de variables. Aunque solo entre los tipos numérico y carácter. Por eso es recomendable el acostumbrarse a utilizar los conversores que incluye PL/SQL.

Funciones de conversión de tipos de PL/SQL y SQL

Funciones	Descripción	Familias que Convierte
TO_CHAR	Convierte su argumento en tipo VARCHAR2, dependiendo del especificador de formato opcional	Numéricos y fechas
TO_DATE	Convierte su argumento en tipo DATE, dependiendo del especificador de formato opcional	Carácter
TO_NUMBER	Convierte su argumento en tipo NUMBER, dependiendo del especificador de formato opcional	Carácter
RAWTOHEX	Convierte un valor RAW en una representación hexadecimal de la cantidad en binario	Raw
HEXTORAW	Convierte una representación hexadecimal en el equivalente binario	Carácter (rep. Hexadecimal)
CHARTOROWID	Convierte una representación de caracteres de un ROWID al formato interno binario	Carácter (form. Rowid)
ROWIDTOCHAR	Convierte una variable interna ROWID al formato externo de 18 caracteres	Rowid

El resto de funciones se ven en el tema 2 con más detalle

11. AMBITO Y VISIBILIDAD DE LAS VARIABLES

El ámbito de una variable es aquella parte del programa en la que se puede acceder a dicha variable, tienen pues un entorno local, ya que solo son visibles en aquella parte del bloque donde se declaran. Si en un bloque anónimo se declara primero una variable llamada **V_NSS** (boolean) y mas adelante se declara otro bloque en el que existe otra variable con el mismo nombre pero de distinto tipo **V_NSS** (integer); dentro del bloque interno solo se verá la del valor entero y en el boque mas externo la booleana

```
DECLARE
        V_bandera int;
        V_NSS boolean;
BEGIN
…
                DECLARE
                        V_NSS integer;
                        …
                BEGIN

                        Aquí solo son visibles las variables V_bandera y
                        V_NSS integer, pero no la variable V_NSS boolean.

                        …
                END;
        …

        Aquí son visibles las variables V_bandera y V_NSS boolean.
END
```

Si el bloque en vez de ser anónimo, le hubiésemos colocado una etiqueta, nos podríamos referir a esa variable:

```
        <etiqueta>.V_NSS
```

12.-ENTRADA/SALIDA

Los programas PL/SQL suelen realizar operaciones específicas sin interactuar con el operador. Sin embargo, existen funciones que nos pueden ayudar a depurar programas y a interactuar con el usuario mostrando datos por pantalla y pidiendo datos al usuario.

ENTRADA DE DATOS.- cuando trabajamos pidiendo datos al usuario es habitual especificar la opción **SET VERIFY OFF** para evitar que el sistema nos muestre el valor que tenía la variable antes y que nos confirme el nuevo valor que toma.

Para pedir datos al usuario se utiliza una variable de substitución, dentro del código fuente del bloque PL/SQL, si esta variable no está inicializada , le pedirá el valor al usuario.

```
<nombre_identificativo>  TIPO :=&variable_a_solicitar;
```

```
Declare

caracter varchar2(1):='&letra';
VV          NUMBER :=&V;
```

SALIDA DE DATOS.- para mostrar una cadena por pantalla podemos utilizar:

```
DBMS_OUTPUT.PUT_LINE(<cadena_caracteres>);
```

Si los datos a mostrar no son cadenas puede utilizarse la función TO_CHAR() para transformarlo y el operador || para concatenar.

El paquete DBMS_OUTPUT implementa una cola, en la cual se van almacenando los mensajes de salida. Si queremos que los mensajes aparezcan por pantalla tenemos que activar la opción SEVER OUTPUT

Ejemplos:

Escribir un bloque pl/sql que escriba el texto 'hola'

```
Begin
dbms_output.put_line('hola');
end;
/
```

Escribir un bloque pl/sql que introduciendo una cantidad de pulgadas por teclado, nos de su equivalente en centímetros. Nota 1pulgada=2.54 centímetros.

```
declare
pulgadas real;
centimetros real;
begin
pulgadas:='&pulgadas';
centimetros:=2.54*pulgadas;
dbms_output.put_line(pulgadas||' pulgadas equivalen a '||centimetros||' centimetros');
end;
/
```

13. ESTRUCTURAS DE CONTROL

En este punto veremos las estructuras de control, que al igual que otros lenguajes posee PL/SQL. Estas estructuras poseen órdenes condicionales y los bucles. Estas estructuras, junto con las variables, proporcionan a PL/SQL su potencia y flexibilidad. Las estructuras que veremos son:

A) - **IF – THEN – ELSE**
   A - I) - Condiciones nulas

B) - **BUCLES**
   B - I) - Bucles simples
   B - II) - Bucles WHILE
   B - III) - Bucles FOR

C) - **GOTO y ETIQUETAS**
   C - I) - Restricciones de GOTO
   C - II) - Etiquetado

D) - **ORDEN NULL**

A) IF – THEN – ELSE

Su sintaxis es:

```
IF <expresión_booleana1> THEN
        Secuencia_de_órdenes1;
[ELSIF  <expresión_booleana2>  THEN
        Secuencia_de_órdenes2;]
…
[ELSE
        Secuencia_de_órdenes3;]

END IF;
```

Donde *<expresiones_booleanas (n)>* es cualquier expresión que de cómo resultado un valor booleano. Las cláusulas ELSIF y ELSE son opcionales y puede haber tantas cláusulas ELSIF como se quiera.

A – I) – Condiciones Nulas

En el punto 7 de este mismo tema, decíamos que era preferible inicializar las variables, siempre y cuando se pueda determinar un valor, ya que si no PL/SQL las inicializaría a NULL. Pues bien, imaginemos que tenemos que hacer un procedimiento para que diga que valor, de los dos que pasamos es mayor o menor:

```
DECLARE
        Num1   NUMBER;
        Num2   NUMBER;
        RSLT   VARCHAR2(7);
BEGIN
        …
        IF Num1 < Num2 THEN
                RSLT :=  'Yes';
        ELSE
                RSLT := 'No';
        END IF;
END;
```

Funcionaria si asignásemos valores como por ejemplo Num1 := 3 y Num2 := 7. Pero supongamos que Num1 := 3 y que Num2 := NULL. La condición (3 < NULL) sera nula, ejecutará la cláusula ELSE, asignando a RSLT el valor de 'No'. Para solucionar esto incluimos una comprobación de nulidad:

```
DECLARE
        Num1   NUMBER;
        Num2   NUMBER;
        RSLT   VARCHAR2(7);

    BEGIN
        …

        IF Num1  IS NULL OR
            Num2  IS NULL  THEN
                RSLT := 'Desconocido';

        ELSIF  Num1 < Num2  THEN
            RSLT :=  'Yes';

        ELSE
            RSLT := 'No';

        END IF;

END;
```

Escribir un bloque pl/sql que permita calcular el precio de un billete de ida y vuelta en avión , conociendo la distancia a recorrer, el número de días de estancia y sabiendo que si la distancia es superior es superior a 1000 km y el numero de días de estancia es superior a 7, la linea aérea le hace un descuento del 30%. El precio por km recorrido es de 8.5 euros.

```
declare
p_des constant real :=0.3;
p_kilo constant real:=8.5;
distancia real;
n_dias real;
descuento real;
p_billete real;
begin
n_dias:='&n_dias';
distancia:='&distancia';
p_billete:=distancia* p_kilo;
if (distancia>1000 and n_dias>7)
  then
    descuento:=p_des*p_billete;
  else
    descuento:=0;
end if;
p_billete:=p_billete-descuento;
dbms_output.put_line('el precio del billete de avión es de '||p_billete||' euros');
end;
/
```

B) – BUCLES

B – I) – Bucles simples

Son el tipo más básico su sintaxis es:

```
LOOP
        <Secuencia_de_órdenes>;

END LOOP;
```

Este bucle seria infinito, no tiene condición de parada. Para salir de un bucle le pondremos la orden EXIT, que su sentencia es:

```
EXIT [ WHEN <Condición> ];
```

Esta orden seria equivalente a:

```
IF <Condición> THEN
        EXIT;
END IF;
```

Escribe un bloque pl/sql que permita calcular la suma de los 1000 primeros números naturales.

```
declare
i integer;
suma real;
begin
  suma:=0;
  i:=1;
  loop
    suma:=suma+i;
    i:=i+1;
    exit when i>1000;
 end loop;
 dbms_output.put_line('la suma de los 1000 primeros numeros naturales es '||suma);
end;
/
```

B – II) – Bucles WHILE

Su sintaxis:

```
WHILE <Condición> LOOP
        <Secuencia_de_órdenes>;
END LOOP;
```

De esta forma, siempre antes de entrar en el bucle evalúa la condición, si es verdadera entrará. Si la condición es falsa o nula el bucle se termina. Por eso hay que tener en cuenta que si la condición del bucle no toma el valor TRUE la primera vez que se le comprueba el bucle no llegará nunca a ejecutarse.

Pueden usarse las órdenes EXIT o EXIT WHEN dentro de un bucle WHILE para salir del bucle, sin llegar a terminar le Condición.

Escribe un bloque pl/sql que permita calcular la suma de los 1000 primeros números naturales.

```
declare
i integer;
suma real;
begin
  suma:=0;
  i:=1;
  while i<=1000 loop
    suma:=suma+i;
    i:=i+1;
 end loop;
 dbms_output.put_line('la suma de los 1000 primeros numeros naturales es '||suma);
end;
/
```

B – III) – Bucles FOR

En el caso en que sepamos el número de iteraciones en que se ejecutarán los bucles simples y while utilizaremos los bucles FOR, su sintaxis es:

```
FOR <Contador_bucle> IN [ REVERSE ] menor … mayor LOOP
        Secuencia_de_órdenes
END LOOP;
```

Donde <*Contador_bucle*> es una variable que no hace falta que se declare ya que lo hace de forma implícita. Los valores *menor…mayor* muestra el rango en que se ejecutará el bucle.

Escribe un bloque pl/sql que permita calcular la suma de los 1000 primeros números naturales.

```
declare
i integer;
suma real;
begin
  suma:=0;
  for i in 1 ..1000 loop
     suma:=suma+i;
  end loop;
  dbms_output.put_line('la suma de los 1000 primeros numeros naturales es '||suma);
end;
/
```

C) – GOTO Y ETIQUETAS

GOTO *Etiqueta*;

```
declare
i integer;
suma real;
begin

  suma:=0;
  goto resultado;
  for i in 1 ..1000 loop
     suma:=suma+i;
  end loop;
  <<resultado>>
  dbms_output.put_line('la suma de los 1000 primeros numeros naturales es '||suma);
end;
/
```

C – I) – Restricciones de GOTO

-**No** puede haber **otra etiqueta** en el entorno actual **con el mismo nombre.**

-La etiqueta debe preceder a un bloque o a un conjunto de órdenes ejecutables.

-**No** se puede saltar al **interior** de un **bucle**

-**No** se puede saltar al **interior** de una **orden IF**

C – II) – Etiquetado

A los bucles pueden ponérseles etiquetas de forma que las usemos en la sentencia EXIT. En el caso de que se le añada una etiqueta a un bucle habrá que ponerla también al final del bucle.

Escribe un bloque pl/sql que permita calcular la suma de los 1000 primeros números naturales.

```
declare
i integer;
suma real;
begin
   suma:=0;
   i:=1;
   <<mi_bucle>>
   while i<=1000 loop
         suma:=suma+i;
                  i:=i+1;
   end loop mi_bucle;
   dbms_output.put_line('la suma de los 1000 primeros numeros naturales es '||suma);
end;
/
```

D) – ORDEN NULL

En algunos casos, podríamos querer indicar explícitamente que no se realice ninguna acción. Esto puede hacerse mediante la orden NULL, que es una orden que no tiene ningún efecto.

Realizar un bloque pl/sql que calcule el valor absoluto de un número negativo.

```
declare
i integer:=&i;
begin
   if i<0
      then
        dbms_output.put_line('el valor absoluto de '||i||' es: '||-i);
      else
        null;
   end if;
end;
/
```

14.-EJERCICIOS PROPUESTOS

1º.- Realizar un bloque Pl/Sql que calcule el importe de una factura sabiendo que el IVA a aplicar es del 12% y que si el importe bruto de la factura es superior a 50.000 € se debe realizar un descuento del 5%.

2º.- Realizar un bloque Pl/Sql en el que introducimos tres números por teclados y nos los visualiza de mayor a menor

3º.- Realizar un bloque Pl/Sql que calcule el salario neto semanal de un trabajador en función del número de horas trabajadas y la tasa de impuestos de acuerdo a las siguientes hipótesis:

- Las primeras 35 horas se pagan a tarifa normal
- Las horas que pasen de 35 se pagan 1.5 veces la tarifa normal
- Las tasas de impuestos son:
  - Los primeros 50 dólares son libres de impuestos
  - Los siguientes 40 dólares tienen un 25% de impuestos
  - Los restantes de 45% de impuestos

4º.- Realizar un bloque pl/sql que calcule las raíces de la ecuación ($ax^2 + bx + c = 0$) teniendo en cuenta los siguientes casos:

a. Utilizaremos la fórmula siguiente: $x= (-b \pm \sqrt{(b^2 - 4ac)})/(2a)$ La expresión $d = b^2 - 4ac$ se denomina discriminante.

a1. Si d>0 entonces hay dos raíces reales

a2. Si d=0 entonces hay una raíz real que es x=-b/(2a)

a3. Si d<0 entonces hay dos raíces complejas de la forma: x+yi, x-yi. Siendo x el valor –b/2a e y de $\sqrt{(|b^2 -4ac|)}/(2a)$

5º.- Realizar un bloque pl/sql que nos pida un valor numérico por teclado y nos diga si dicho número es primo o no. Un número es primo si solo es divisible por el mismo y por la unidad.

6º.- Realizar un bloque pl/sql que nos pida 2 valores numéricos enteros positivos por teclado y calcule la multiplicación de dichos números por sumas sucesivas.

7º.- Realizar un bloque pl/sql que nos pida dos números enteros positivos por teclado y calcule su división por restas sucesivas.

8º.- Realizar un bloque pl/sql que nos pida un numero entero por teclado y calcule su factorial.

9º.- Realizar un bloque pl/sql que calcule la suma de los números enteros hasta un número dado introducido por el usuario

10º.- Realizar un bloque pl/sql que calcule las tablas de multiplicar del 1 al 9