

10.1 INTRODUCCION.

Hasta el momento, hemos trabajado con la base de datos de manera interactiva. Esta forma de trabajar, implica que en un entorno de producción todos los usuarios **deberían conocer y manejar SQL**.

Para solucionar esto, Oracle incorpora un gestor PL/SQL en el servidor de la base de datos. Este lenguaje, basado en el lenguaje ADA, incorpora todas las características de los lenguajes de tercera generación: manejo de variables, estructura modular (procedimientos y funciones), estructuras de control (bifurcaciones, bucles y demás de estructuras) , control de excepciones. También incorpora un soporte para la POO, por lo que se considera un lenguaje procedimental y orientado a objetos.

Los programas creados con PL/SQL se pueden almacenar en la base de datos como cualquier otro objeto quedando así disponibles para su ejecución por los usuarios.

Por todo ello, el conocimiento de este lenguaje es imprescindible para poder trabajar en el entorno Oracle, tanto para administradores de la base de datos como para desarrolladores de aplicaciones.

Los tipos básicos de un programa PL/SQL son: bloques anónimos, procedimientos y funciones.

10.2 CARACTERISTICAS DEL LENGUAJE.

PL/SQL es un lenguaje procedimental diseñado por Oracle para trabajar con la base de datos.

La unidad de trabajo es el **bloque**, constituido por un conjunto de declaraciones, instrucciones y mecanismos de gestión de errores y excepciones.

Tiene tres zonas claramente definidas:

- Una **zona de declaraciones** donde se declaran objetos locales (variables, constantes, etc). Suele ir precedida por la cláusula DECLARE (o IS/AS en los procedimientos y funciones). Es opcional.
- Un **conjunto de instrucciones** precedido por la cláusula BEGIN.
- Una zona de tratamiento de excepciones precedido por la cláusula EXCEPTION. Esta zona, igual que la de declaraciones, es opcional.

Formato genérico del bloque es:

```
[ DECLARE
  <declaraciones> ]
BEGIN
  <instrucciones>
[EXCEPTION
  <gestión de excepciones> ]
END;
```

Ejemplo

En el siguiente ejemplo se borra el departamento número 20, pero evitando posibles errores por violar restricciones de integridad referencial, pues el departamento tiene empleados y hay un historial de trabajos asociados. Para ello crearemos antes un departamento provisional, al que asignamos los empleados y el historial de trabajo del departamento 20 antes de borrar dicho departamento. El programa también informa del número de empleados afectados.

```

DECLARE
v_num_empleados NUMBER(2);
BEGIN
INSERT INTO departamentos VALUES (99,'Human Resources',203,2400);
UPDATE empleados SET departamento_id=99
WHERE departamento_id=20;
UPDATE trabajo_historial SET departamento_id=99
WHERE departamento_id=20;
v_num_empleados := SQL%ROWCOUNT;
DELETE FROM departamentos
WHERE departamento_id=20;
DBMS_OUTPUT.PUT_LINE(v_num_empleados || 'empleados ubicados en provisional');
EXCEPTION
  WHEN OTHERS THEN
    RAISE_APPLICATION_ERROR(-20000, 'error en aplicacion');
END;
/

```

Utilizando SQLDeveloper no hace falta poner / al final. Y añadiremos antes de EXCEPTION la orden **COMMIT** que permite realizar la actualización de forma inmediata sobre la base de datos que esta conectada.

La utilización de bloques supone una notable mejora de rendimiento ya que se envían los bloques completos al servidor para que sean procesados, en lugar de cada sentencia SQL. Así se ahorran muchas operaciones de E/S.

Los bloques PL/SQL se pueden anidar para conseguir distintas funcionalidades (por ejemplo, para evitar la propagación de excepciones) como veremos más adelante. A estos bloques se les llama **bloques anidados**.

```

DECLARE
...
BEGIN
...
  DECLARE -- comienzo del bloque interior anidado
    ...
    BEGIN
      ...
      EXCEPTION
      ...
      END; -- fin del bloque interior anidado
..
EXCEPTION
...
END

```

10.3 DEFINICION DE DATOS COMPATIBLES CON SQL.

- Declaración e inicialización de variables

Variable representa una dirección en memoria del ordenador donde se almacena un dato que puede ser recuperado y modificado a lo largo del programa. Antes de utilizar una variable hay que declararla .

Todas las variables PL/SQL deben declararse en la sección declarativa antes de su uso.

El formato genérico para declarar una variable es el siguiente:

<nombre_de_variable> <tipo> [NOT NULL] [{:=<valor>}; DEFAULT}

La opción DEFAULT (o bien la asignación «:=») sirve para asignar valores por defecto a la variable desde el momento de su creación.

DECLARE

```
importe NUMBER(8,2);
contador NUMBER(2,0) := 0 ;
nombre CHAR(20) NOT NULL := 'MIGUEL';
```

Para cada variable se debe especificar el tipo. No se puede, como en otros lenguajes, indicar una lista de variables del mismo tipo y, a continuación, el tipo. PL/SQL no lo permite.

La opción NOT NULL fuerza a que la variable tenga siempre un valor. Si se usa, deberá inicializarse la variable en la declaración con DEFAULT o con := para la inicialización.

Por ejemplo:

```
nombre CHAR(20) NOT NULL DEFAULT 'MIGUEL';
```

También se puede inicializar una variable que no tenga la opción NOT NULL. Por ejemplo:

```
nombre CHAR(20) DEFAULT 'MIGUEL';
```

Si no se inicializan las variables en PL/SQL, se garantiza que su valor es NULL.

PL/SQL dispone de tipos de datos compatibles con los tipos utilizados para las columnas de las tablas: NUMBER, VARCHAR2, DATE, etcétera, además de otros propios, como BOOLEAN.

Boolean

Representan los valores lógicos verdadero, falso y nulo. Se representan mediante las palabras TRUE, FALSE y NULL escritas directamente en el código y sin comillas. Se pueden utilizar para asignar valores a variables y constantes, o para comprobar el resultado de expresiones lógicas.

Ejemplos: `v_cobrado := TRUE;` `v_enviado := FALSE;`

Fecha/hora

Se utilizan para representar valores de fecha y hora según los distintos formatos estudiados anteriormente. Se representan mediante la expresión que indica el tipo DATE o TIMESTAMP seguida de la cadena delimitada que indica el valor que queremos representar.

Por ejemplo: `DATE '2005-11-09'` `TIMESTAMP '2005-11-09 13:50:00'.`

Las declaraciones de los datos deben realizarse en la sección de declaraciones.

```
DECLARE
  Importe NUMBER(8,2);
  Nombre char(20);
  Fecha date:= current_date;
  Mayoreadad boolean := true;
```

```
BEGIN
```

```
...
```

• **Uso de los atributos %TYPE y %ROWTYPE**

En lugar de indicar explícitamente el tipo y la longitud de una variable existe la posibilidad de utilizar los atributos %TYPE y %ROWTYPE para declarar variables que sean del mismo tipo que otros objetos ya definidos.

- %TYPE declara una variable del mismo tipo que otra, o que una columna de una tabla.

Su formato es: nombre_variable nombre_obj eto%TYPE;

- %ROWTYPE declara una variable de registro cuyos campos se corresponden con las columnas de una tabla o vista de la base de datos.

Su formato es: nombre_variable nombre__objeto%ROWTYPE;

Ejemplos:

```
DECLARE
  Importe NUMBER(8,2);
  Nombre char(20);
```

```
Create table clientes
( dni int primary key,
  nombre varchar(20),
  direccion varchar(30));
```

Declara la variable total del mismo tipo que la variable importe que se habrá definido previamente.

```
total importe%TYPE; // total es de tipo numero(8,2)
```

Declara la variable nombre_moroso del mismo tipo que La columna nombre de la tabla clientes.

- nombre_moroso clientes.nombre%TYPE; // nombre_moroso es de tipo varchar(20)

Declara la variable moroso que podrá contener una fila de la tabla clientes.

- moroso clientes%ROWTYPE; (moroso variable formado por 3 campos dni, nombre, direccion)

Para hacer referencia a cada uno de los campos indicaremos el nombre de la variable, un punto y el nombre del campo que coincide con el de la columna correspondiente.

Por ejemplo:

moroso.dni o moroso.direccion

- Constantes

Constante representan datos almacenados en memoria, con la salvedad de que éstos no pueden ser modificados a lo largo de la ejecución de un programa.

Las declaramos con el siguiente formato:

Aunque PL/SQL no diferencia entre mayúsculas y minúsculas en los identificadores, sí lo hace en los literales y con el contenido de las variables y constantes.

`<nombre_de_constante> CONSTANT <tipo> := <valor>;`

Por ejemplo:

`Pct_iva CONSTANT REAL := 16;`

- Operadores

Los **operadores** se utilizan para asignar valores y formar expresiones. PL/SQL dispone de operadores de:

- **Asignación.** `:=` Asigna un valor a una variable.
Por ejemplo: `Edad := 19;`
- **Concatenación.** `||` Une dos o más cadenas.
Por ejemplo: `'buenos' || 'días'` dará como resultado `'buenosdías'`.
- **Comparación.** `=`, `!=`, `<`, `>`, `<=`, `>=`, `in`, `is null`, `like`, `between`,... Funcionan igual que en SQL.
- **Aritméticos.** `+`, `-`, `*`, `/`, `**`. Se emplean para realizar cálculos. Algunos de ellos se pueden utilizar también con fechas.
- **Lógicos.** `AND`, `OR` y `NOT`. Permiten operar con expresiones que devuelven valores booleanos (comparaciones, etcétera). A continuación, se detallan las tablas de valores de los operadores lógicos `AND`, `OR` y `NOT`, teniendo en cuenta todos los posibles valores, incluida la ausencia de valor (`NULL`).

Otros indicadores y delimitadores.

`()` Delimitador de expresiones. Delimitador de literales de cadena.
 Delimitador de identificadores (utilización desaconsejable, en general).
`«»` Etiquetas.
`/* */` Delimitador de comentarios de varias líneas.
 Indicador de comentario de una línea.
`%` Indicador de atributo (`TYPE`, `ROWTYPE`, `FOUND`,...).
`:` Indicador de variables de transferencia (`bind`).
 Separador de ítem de lista.
 Terminador de instrucción.
`@` Indicador de enlace de base de datos.

Los comentarios se utilizan para documentar datos generales y particulares de los programas e incluso para añadir legibilidad y organización a nuestros programas: líneas de separación, etcétera. En PL/SQL, podemos insertar comentarios en cualquier parte del programa.

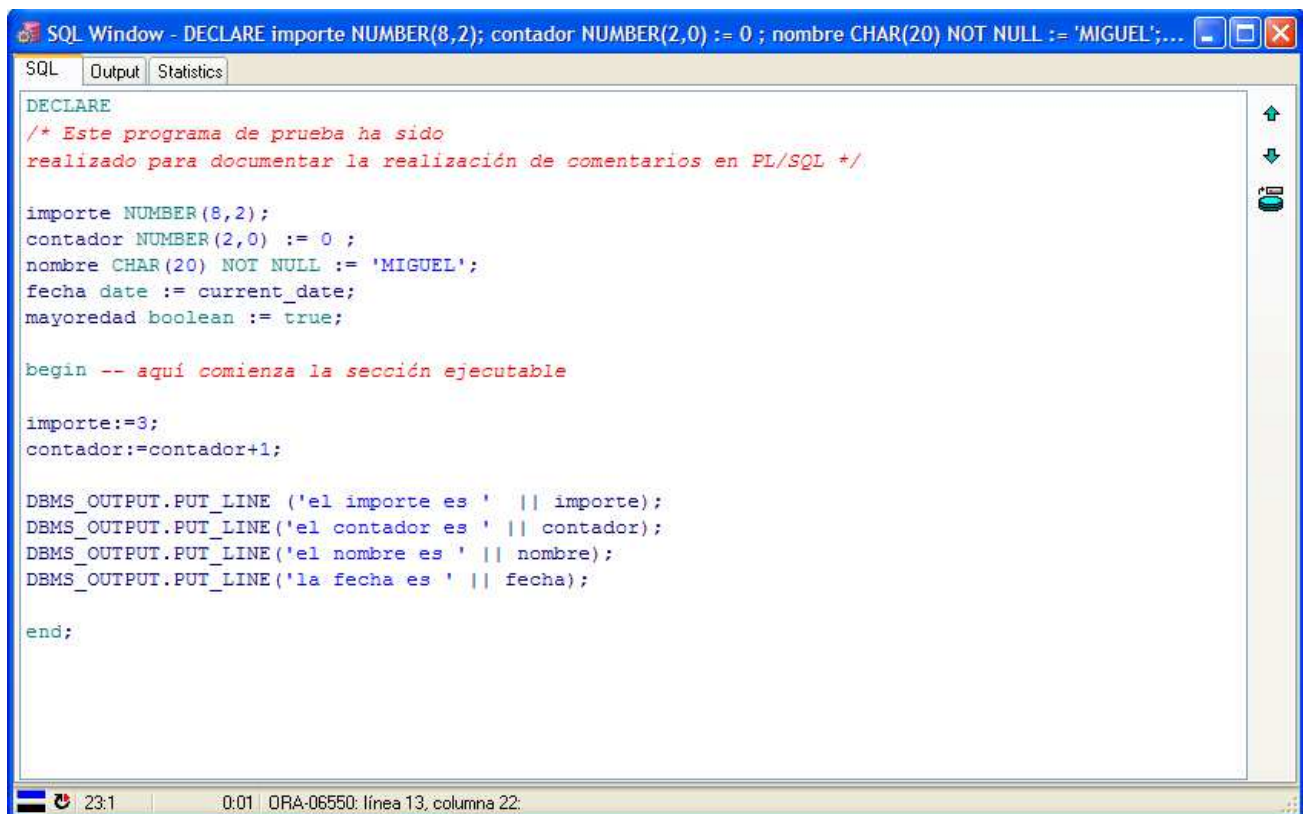
Estos comentarios pueden ser:

- De línea con "--". Todo lo que le sigue en esa línea será considerado comentario.
 - Todos los comentarios incluidos en el código hasta el momento son de este tipo.
- De varias líneas con "/*" <comentarios> "*/" (igual que en C). Se pueden incluir // en cualquier sección del programa. Es aconsejable, aunque no obligatorio, que incluyan una o varias líneas completas.

```
/* Este programa de prueba ha sido realizado para documentar la realización de
comentarios en PL/SQL */
```

```
BEGIN -- aquí comienza la sección ejecutable
```

Ejemplo 1

The image shows a screenshot of an 'SQL Window' application. The title bar reads 'SQL Window - DECLARE importe NUMBER(8,2); contador NUMBER(2,0) := 0 ; nombre CHAR(20) NOT NULL := 'MIGUEL';...'. The window has three tabs: 'SQL', 'Output', and 'Statistics', with 'SQL' being the active tab. The main text area contains the following PL/SQL code:

```
DECLARE
/* Este programa de prueba ha sido
realizado para documentar la realización de comentarios en PL/SQL */

importe NUMBER(8,2);
contador NUMBER(2,0) := 0 ;
nombre CHAR(20) NOT NULL := 'MIGUEL';
fecha date := current_date;
mayoread boolean := true;

begin -- aquí comienza la sección ejecutable

importe:=3;
contador:=contador+1;

DBMS_OUTPUT.PUT_LINE ('el importe es ' || importe);
DBMS_OUTPUT.PUT_LINE ('el contador es ' || contador);
DBMS_OUTPUT.PUT_LINE ('el nombre es ' || nombre);
DBMS_OUTPUT.PUT_LINE ('la fecha es ' || fecha);

end;
```

The status bar at the bottom shows a cursor icon, the text '23:1', a separator, and then '0:01 | ORA-06550: línea 13, columna 22:'. On the right side of the code editor, there are three vertical icons: an upward arrow, a downward arrow, and a circular arrow.

10.4 ESTRUCTURAS DE CONTROL.

Las estructuras de control de PL/SQL son las habituales de los lenguajes de programación estructurados: IF, CASE, WHILE, FOR Y LOOP.

Estructuras de control		
Alternativa simple IF <condición> THEN instrucciones; END IF; Alternativa doble IF <condición> THEN instrucciones; ELSE instrucciones ; END IF;	Alternativa múltiple (elsif) IF <condición> THEN instrucciones; ELSIF <condición2> THEN instrucciones; ELSIF <condición3> THEN instrucciones; ELSE instrucciones ; END IF;	Alternativa múltiple (case) CASE [<expresión>] WHEN <test1> THEN <instrucciones1>; WHEN <test2> THEN <instrucciones2>; WHEN <test3> THEN <instrucciones3>; [ELSE <otrasinstruccio-nes>;] END CASE;
Estructuras de control repetitivas		
Mientras WHILE <condición> LOOP Instrucciones ; END LOOP;	Para FOR <variable> IN [REVERSE] <mínimo>.. máximo LOOP instrucciones; END LOOP;	Iterar... fin iterar salir si... LOOP instrucciones ; EXIT WHEN <condición>; instrucciones; END LOOP;

SENTENCIAS CONDICIONALES

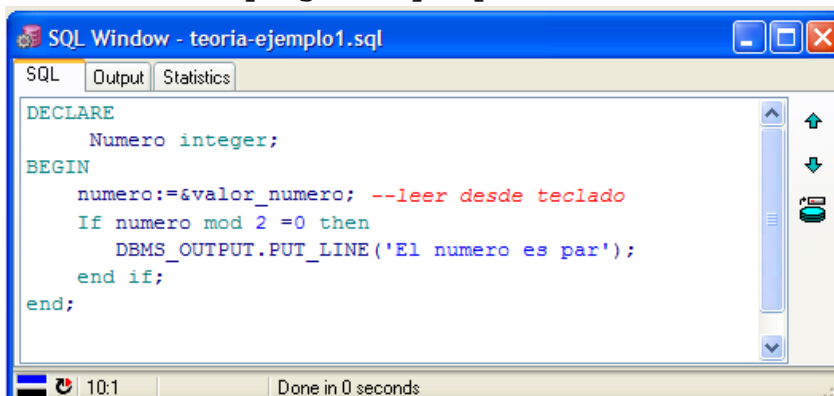
Alternativa simple

La sentencia solo se ejecuta si se cumple la condición. En caso contrario el programa sigue su curso sin ejecutar la sentencia.

```
IF <condición>
THEN
instrucciones;
END IF;
```

Ejemplo 2

Realiza un programa que pida un número e indique si es par.



```
SQL Window - teoria-ejemplo1.sql
SQL Output Statistics
DECLARE
    Numero integer;
BEGIN
    numero:=&valor_numero; --leer desde teclado
    If numero mod 2 =0 then
        DBMS_OUTPUT.PUT_LINE('El numero es par');
    end if;
end;
```

10:1 Done in 0 seconds

Notas:

DBMS_OUTPUT.PUT_LINE() -> nos permite mostrar los datos que se pone entre los paréntesis en el monitor. – ESCRITURA

&variable -> nos permite leer desde el teclado y lo guarda en la variable.

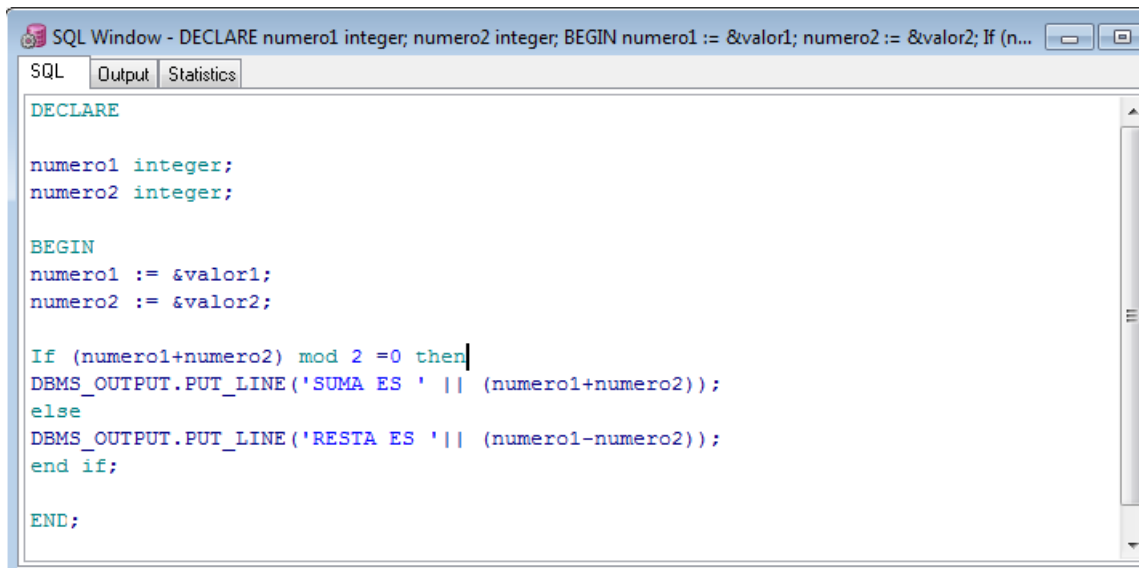
Alternativa doble

Si se cumple la condición ejecutará la **sentencia1**, sino ejecutará la **sentencia2**. En cualquier caso, el programa continuará a partir de la **sentencia2**.

```
IF <condición>
THEN instrucciones;
ELSE
instrucciones ;
END IF;
```

Ejemplo 3

Realizar un programa que pida dos números enteros. Si la suma de los dos es par se visualizará en pantalla la suma de ambos. Si no es así, se dará el resultado de restar el número mayor al menor.



```
SQL Window - DECLARE numero1 integer; numero2 integer; BEGIN numero1 := &valor1; numero2 := &valor2; If (n...
SQL Output Statistics
DECLARE
numero1 integer;
numero2 integer;
BEGIN
numero1 := &valor1;
numero2 := &valor2;
If (numero1+numero2) mod 2 =0 then
DBMS_OUTPUT.PUT_LINE('SUMA ES ' || (numero1+numero2));
else
DBMS_OUTPUT.PUT_LINE('RESTA ES ' || (numero1-numero2));
end if;
END;
```

Alternativa múltiple (elsif)

Con este formato el flujo del programa únicamente entra en una de las condiciones. Si una de ellas se cumple, se ejecuta la sentencia correspondiente y salta hasta el final de la estructura para continuar con el programa.

```
IF <condición> THEN instrucciones;
ELSIF <condición2> THEN
instrucciones;
ELSIF <condición3> THEN
instrucciones;
ELSE
instrucciones ;
END IF;
```


Ejemplo 4

Se pide a un alumno que introduzca una nota. Si la nota es mayor que cero y menor que 5 además de escribir la nota aparecerá la palabra **cate**. Si está entre 5 y 10 además de escribir la nota la palabra **aprobado**. Si se introduce otro número se indicará que eso no es una nota.

```

SQL Window - DECLARE nota integer; BEGIN nota:=&valor; IF nota>0 and nota<5 then DBMS_...
SQL Output Statistics
DECLARE
nota integer;

BEGIN
    nota:=&valor;

    IF nota>0 and nota<5 then
        DBMS_OUTPUT.PUT_LINE('La nota es ' || nota || ' CATE');
    ELSIF nota=5 and nota<=10 then
        DBMS_OUTPUT.PUT_LINE('La nota es ' || nota || ' APROBADO');
    ELSE
        DBMS_OUTPUT.PUT_LINE('El numero introducido no es una nota');
    END IF;
END;

15:5 Done in 0 seconds

```

Alternativa múltiple (case)

```

CASE [<expresión>]
WHEN <test1> THEN
    <instrucciones1>;
WHEN <test2> THEN
    <instrucciones2>;
WHEN <test3> THEN
    <instrucciones3>;
[ELSE <otrasinstruccio-nes>;]
END CASE;

```

Ejemplo 5

Realizar un programa que pida al usuario tres numeros y a continuación ofrezca las siguientes 3 opciones:

- 1.- Suma de los tres numeros.
- 2.- Producto de los tres números.
- 3.- Promedio.

```

SQL Window - teoria-ejemplo5.sql
SQL Output Statistics
DECLARE
n1 integer;
n2 integer;
n3 integer;
op integer;

BEGIN
    n1:=&v1;
    n2:=&v2;
    n3:=&v3;

    DBMS_OUTPUT.PUT_LINE('1.- Suma de los tres numeros. ');
    DBMS_OUTPUT.PUT_LINE('2.- Producto de los dos primeros números. ');
    DBMS_OUTPUT.PUT_LINE('3.- Promedio. ');

    op:=&v4;
    DBMS_OUTPUT.PUT_LINE('La opción seleccionada es ' || op);
    CASE op
    WHEN 1 THEN DBMS_OUTPUT.PUT_LINE('La suma es ' || (n1+n2+n3));
    WHEN 2 THEN DBMS_OUTPUT.PUT_LINE('El producto es ' || (n1*n2));
    WHEN 3 THEN DBMS_OUTPUT.PUT_LINE('Promedio' || ((n1+n2+n3)/3));
    ELSE DBMS_OUTPUT.PUT_LINE('Elección no correcta. ');
    END CASE;
END;

21:66 SQL script saved successfully

```

ESTRUCTURAS DE CONTROL REPETITIVAS

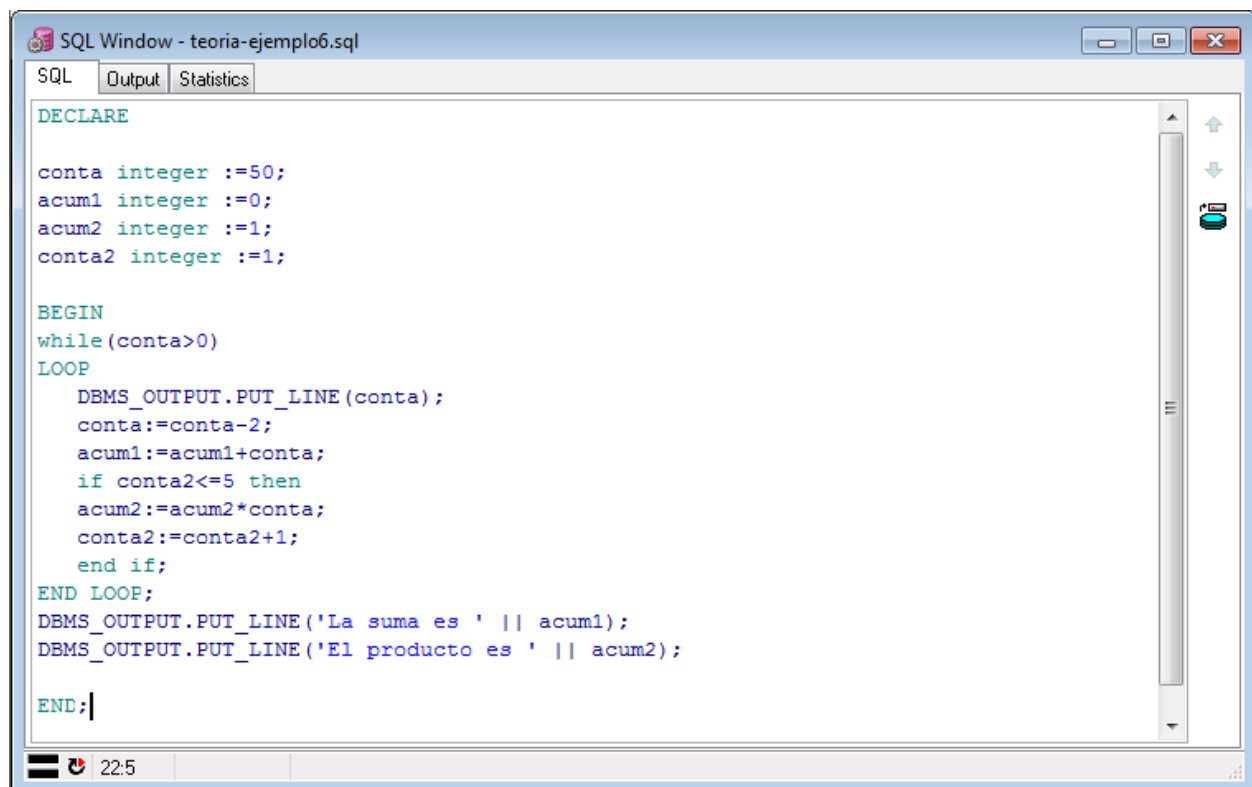
Los bucles son estructuras que permiten ejecutar partes del código de forma repetida mientras se cumpla una condición. Esta condición puede ser simple o compuesta de otras condiciones unidas por operadores lógicos

Mientras

```
WHILE <condición>
LOOP
Instrucciones ;
END LOOP;
```

Ejemplo 6

Programa que imprima los números pares y al final imprima la suma de los 50 primeros números pares y el producto de los 5 primeros números pares.

A screenshot of a SQL Window titled "teoria-ejemplo6.sql". The window has tabs for "SQL", "Output", and "Statistics". The SQL tab is active, showing a PL/SQL program. The program declares four integer variables: conta (50), acum1 (0), acum2 (1), and conta2 (1). It then enters a while loop that continues as long as conta is greater than 0. Inside the loop, it prints the current value of conta, decrements it by 2, adds it to acum1, and if conta2 is less than or equal to 5, it multiplies acum2 by conta and increments conta2. After the loop, it prints the final sum (acum1) and product (acum2). The status bar at the bottom shows a refresh icon, a cursor icon, and the time 22:5.

```
SQL Window - teoria-ejemplo6.sql
SQL Output Statistics
DECLARE
conta integer :=50;
acum1 integer :=0;
acum2 integer :=1;
conta2 integer :=1;

BEGIN
while(conta>0)
LOOP
    DBMS_OUTPUT.PUT_LINE(conta);
    conta:=conta-2;
    acum1:=acum1+conta;
    if conta2<=5 then
        acum2:=acum2*conta;
        conta2:=conta2+1;
    end if;
END LOOP;
DBMS_OUTPUT.PUT_LINE('La suma es ' || acum1);
DBMS_OUTPUT.PUT_LINE('El producto es ' || acum2);

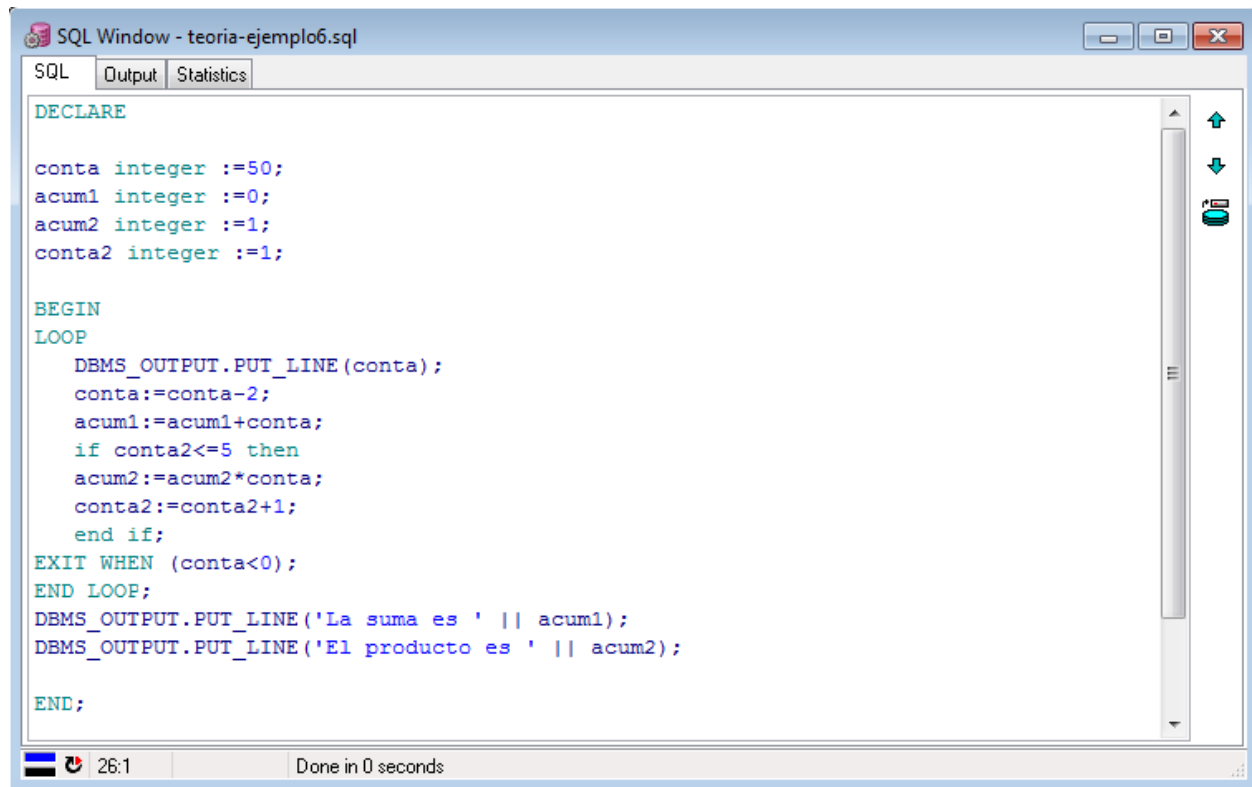
END;
```

Iterar... fin iterar salir si...

```
LOOP
    instrucciones ;
EXIT WHEN <condición>;
    instrucciones;
END LOOP;
```

Ejemplo 7

Programa que imprima los números pares y al final imprima la suma de los 50 primeros números pares y el producto de los 5 primeros números pares.



```

SQL Window - teoria-ejemplo6.sql
SQL Output Statistics

DECLARE

conta integer :=50;
acum1 integer :=0;
acum2 integer :=1;
conta2 integer :=1;

BEGIN
LOOP
  DBMS_OUTPUT.PUT_LINE(conta);
  conta:=conta-2;
  acum1:=acum1+conta;
  if conta2<=5 then
    acum2:=acum2*conta;
    conta2:=conta2+1;
  end if;
EXIT WHEN (conta<0);
END LOOP;
DBMS_OUTPUT.PUT_LINE('La suma es ' || acum1);
DBMS_OUTPUT.PUT_LINE('El producto es ' || acum2);

END;

26:1 Done in 0 seconds

```

Para

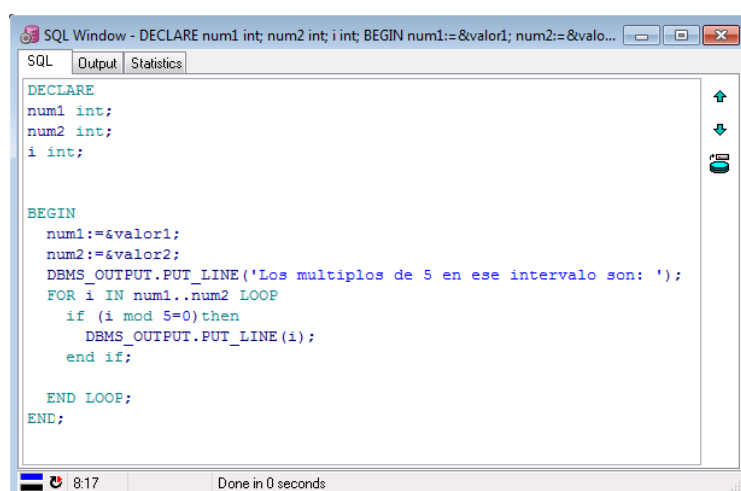
```

FOR <variable> IN <mínimo>..<máximo>
LOOP
  instrucciones;
END LOOP;

```

Ejemplo 8

Programa que escriban los múltiplos de 5 que hay entre dos números introducidos.



```

SQL Window - DECLARE num1 int; num2 int; i int; BEGIN num1:=&valor1; num2:=&valo...
SQL Output Statistics

DECLARE
num1 int;
num2 int;
i int;

BEGIN
  num1:=&valor1;
  num2:=&valor2;
  DBMS_OUTPUT.PUT_LINE('Los multiplos de 5 en ese intervalo son: ');
  FOR i IN num1..num2 LOOP
    if (i mod 5=0) then
      DBMS_OUTPUT.PUT_LINE(i);
    end if;
  END LOOP;
END;

8:17 Done in 0 seconds

```