GUÍA DE LABORATORIO 2

PL/SQL

- **Bloque Anónimo**
- **Unidades Léxicas**
- **Tipos de Datos**
- **Salida por Pantalla**
- **Estructuras de Control**
- **Cursores**

¿Qué es PL/SQL?

Es un lenguaje procedimental extensión de SQL, es decir que tenemos todas las prestaciones del lenguaje de SQL (INSERT, UPDATE, SELECT, DELETE) pero de forma estructurada.
Un lenguaje procedimental es aquel que se encarga de efectuar las órdenes paso a paso, es decir que nosotros le podemos dar una lógica ya sobre escrita.
En PL/SQL tenemos unidades de trabajo reutilizables llamadas bloques, al colocar un nombre a estos bloques, podemos crear procedimientos almacenados y funciones.

Bloque Anónimo.

Estructura básica de un programa de la base de datos en PL/SQL; un bloque anónimo se utiliza para ejecutar acciones en la BD. Un bloque anónimo nunca se va a guardar (su estructura/código en la BD), solo se ejecuta.

```
2
3  DECLARE
4     --declaración de variables
5  BEGIN
6     --programación
7  EXCEPTION (opcional)
8     --control de errores
9  END
```

Cuando escribimos un bloque el PL/SQL, utilizamos un conjunto específico de caracteres. El conjunto de caracteres soportado es el siguiente:

- Mayúsculas y minúsculas A…Z y a…z.
- Números 0…9.
- Tabulaciones, espacios y retornos de carro.

- Símbolos ( ) + - * / < > = ¡ ~ ; : . ` @ % , " # $ ^ & _ | { } ¿ [ ]

PL/SQL NO ES "CASE SENSITIVE" EXCEPTO PARA UN PAR DE CASOS. ¿QUÉ SIGNIFICA QUE NO SEA CASE SENSITIVE?

Unidades Léxicas.

Una sentencia de PL/SQL contiene grupos de caracteres, llamados Unidades Léxicas, las cuales se clasifican de la siguiente forma:

1. **Delimitadores:** un delimitador es un símbolo simple o compuesto, que tiene un significado especial en PL/SQL.

SÍMBOLOS SIMPLES		SÍMBOLOS COMPUESTOS	
Operador	**Significado**	**Operador**	**Significado**
+	*Operador de suma*	******	*Operador de exponenciación*
%	*Indicador de atributo*	**< >**	*Operador relacional*
`	*Carácter delimitador de String*	**!=**	*Operador relacional*
.	*Selector*	**~=**	*Operador relacional*
/	*División*	**<=**	*Operador relacional*
( )	*Expresión o delimitador de lista*	**>=**	*Operador relacional*
:	*Indicador de variable host*	**:=**	*Operador de asignación*
,	*Separador de Ítems*	**=>**	*Operador de asociación*
*****	*Multiplicación*	**..**	*Operador de rango*
"	*Delimitador de identificadores*	**||**	*Operador de concatenación*
=	*Operador relacional*	**<<**	*(Comienzo) delimitador de etiqueta*
<	*Operador relacional*	**>>**	*(Fin) delimitador de etiqueta*
>	*Operador relacional*	**--**	*Comentario una sola línea*
@	*Indicador de acceso remoto*	**/***	*(Comienzo) comentario varias líneas*
;	*Terminador de sentencia*	***/**	*(Fin) comentario varias líneas*
-	*Resta/Operador de negación*		

2. **Identificadores:** los identificadores se utilizan para dar nomenclatura a unidades e items de un programa PL/SQL, el cual puede incluir constantes, variables, excepciones, cursores, cursores con variables, subprogramas y packages. Un identificador consiste en una letra seguida, de manera opcional, de más letras, números, signos de dólar, underscores, y signos numéricos. Algunos caracteres como *% - / y espacios* son ilegales. La longitud de un identificador no puede exceder los 30 caracteres.

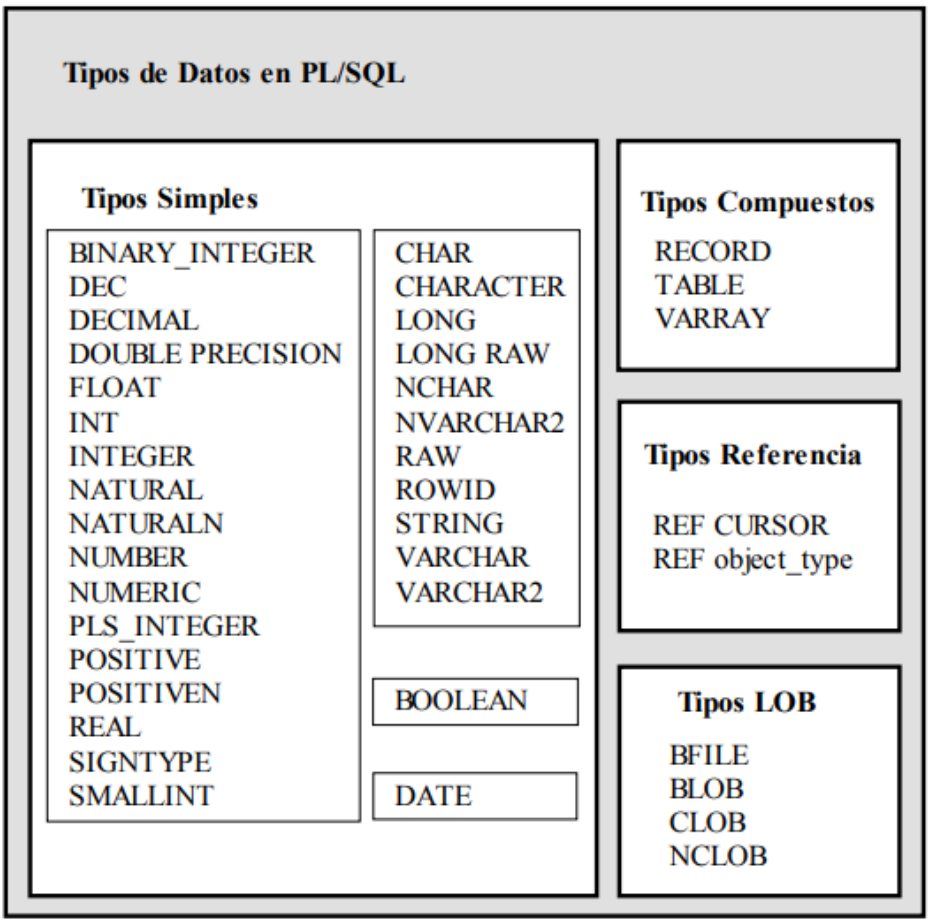
- **Palabras reservadas.**
  Algunos identificadores, llamados Palabras Reservadas, tienen un significado sintáctico especial para PL/SQL, y no pueden ser redefinidas; un claro ejemplo son las palabras BEGIN y END.
  Las palabras reservadas se suelen poner en mayúsculas, para facilitar la lectura del código fuente,
- **Identificadores predefinidos.**
  Los identificadores globales declarados en el package STANDAR, como por ejemplo la excepción INVALID_NUMBER, pueden ser redeclarados… sin embargo, la declaración de identificadores predefinidos es un error, puesto que las declaraciones locales prevalecen sobre las globales.
- **Identificadores con comillas dobles.**
  Por Flexibilidad, PL/SQL permite incluir identificadores con dobles comillas. Estos identificadores no son necesarios muy a menudo, pero a veces pueden ser de gran ayuda. Pueden contener cualquier secuencia de caracteres, incluyendo espacios, pero excluyendo las comillas dobles.

3. **Literales:** un literal es un valor explícito de tipo numérico, carácter, string o booleano, no representado por un identificador. El literal numérico 147, y el literal booleano FALSE, son ejemplos de esto.

   - **Literales numéricos.**
     Enteros y reales.
   - **Literales de tipo carácter.**
     Es un carácter individual entre comillas simples. PL/SQL es 'case sensitive' con los literales de tipo carácter.
   - **Literales de tipo String.**
     Es una secuencia de cero o más caracteres delimitados por comillas simples. Si deseamos que la cadena de caracteres tenga una comilla simple, lo que debemos hacer es repetir la comilla simple. PL/SQL es 'case sensitive' para los literales de tipo String.
   - **Literales de tipo Booleano.**
     Los literales de tipo Booleano son los valores predeterminados TRUE, FALSE, y NULL en el caso de no tener ningún valor. Recordemos siempre que los literales de tipo Booleano son valores y no Strings.

4. **Comentarios.**

Tipos de Datos.



Declaración de variables y asignación de valores a una variable.

Al declarar una variable o constante, podemos darle un valor inicial. Incluso podemos asignarle expresiones, como en el siguiente ejemplo:

```
12  pi REAL := 3.14159;
13  radio REAL := 1;
14  area REAL  := pi*radio*2;
```

Por defecto, las variables se inicializan a NULL, así que las siguientes dos declaraciones serían equivalentes:

```
16  cumple DATE;
17  cumple DATE := NULL;
```

Cuando declaremos una constante, la palabra clave CONSTANT debe preceder a la especificación del tipo.

```
19 limite_de_credito CONSTANT REAL := 250.000;
```

Se utiliza DEFAULT para las variables que tienen un valor típico, mientras que el operador de asignación, se usa en aquellos casos en que las variables no tienen dicho valor, como por ejemplo en contadores y acumuladores.

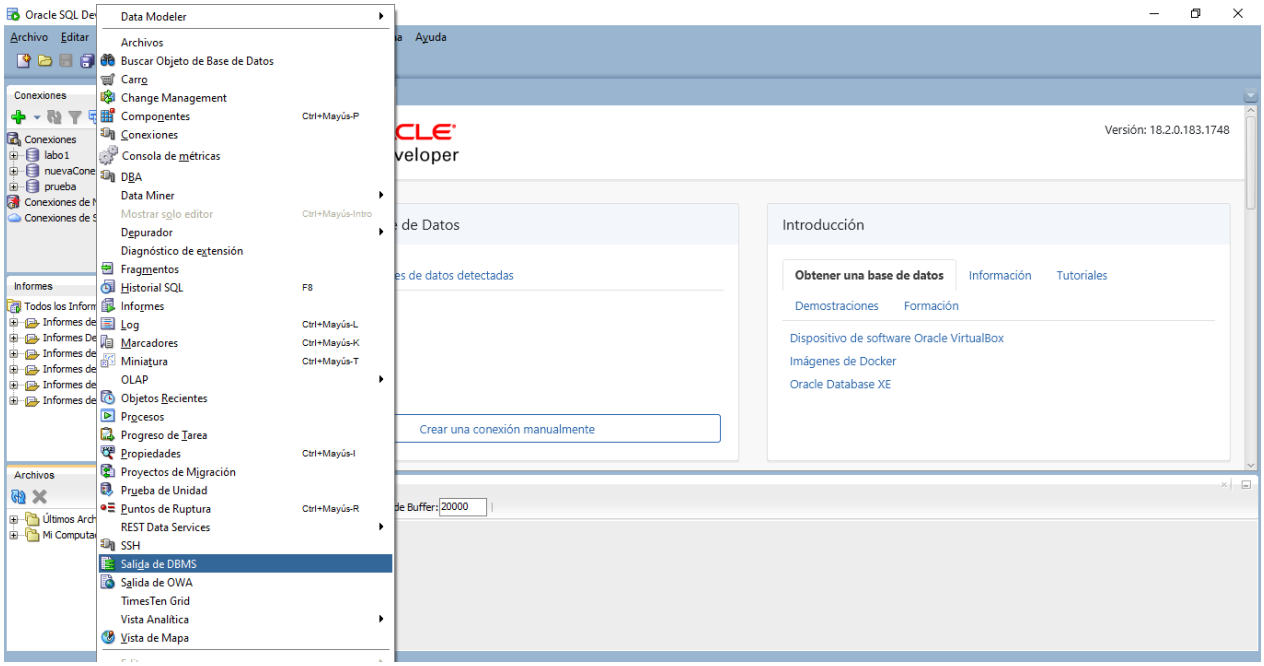
```
21 tipo_sangre CHAR DEFAULT 'O';
22 valido BOOLEAN DEFAULT FALSE;
```

Salida por Pantalla (DBMS OUTPUT)

PL/SQL no posee, como tal, una instrucción "print" ya que está pensado para trabajar con datos, generar los datos para ser leídos por otra herramienta o modificar información en la BD. No es un "lenguaje visual".

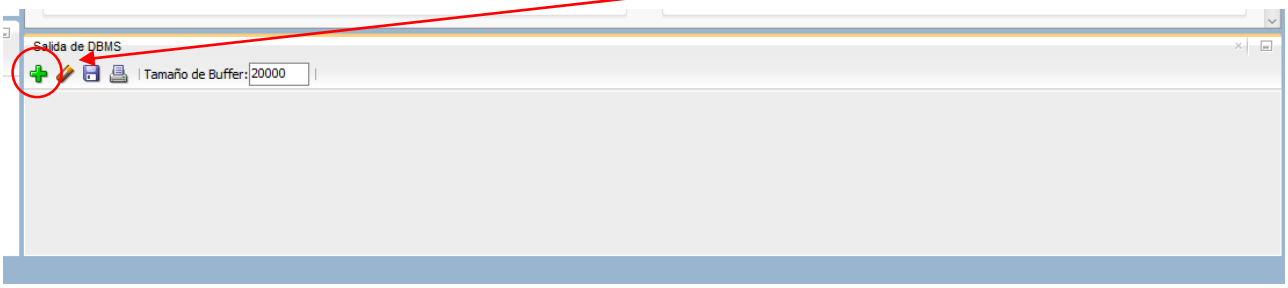
Para solucionar dicha carencia, utilizaremos un paquete PL/SQL y realizaremos los siguientes pasos:

1. Habilitar la salida mediante el comando: SET SERVEROUTPUT ON o podemos activar la ventana de la siguiente forma:

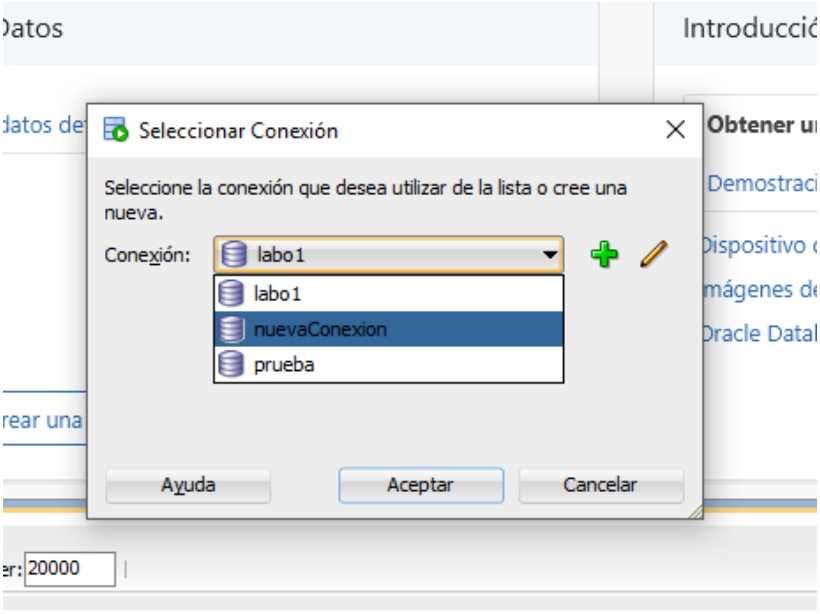


(Ver > Salida DBMS)

Aparecerá la siguiente ventana y haremos clic en la cruz de color verde:



Seleccionamos una conexión y damos clic en aceptar:



2. Escribimos DBMS_OUTPUT.PUT_LINE("<lo que queremos mostrar>");
3. Ejecutamos nuestro bloque.

```
24 BEGIN
25    dbms_output.put_line("Hello World");
26 END
```

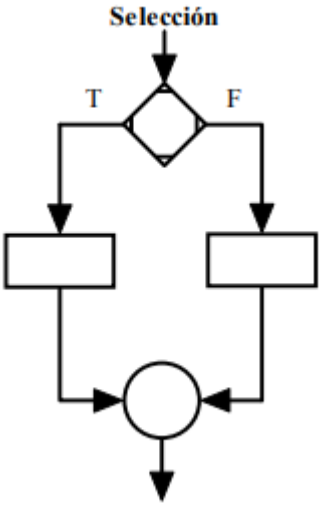
Guardar a partir de un SELECT una variable.

```
30 SELECT <lo que se va a guardar> INTO <variable> FROM <tabla>;
```

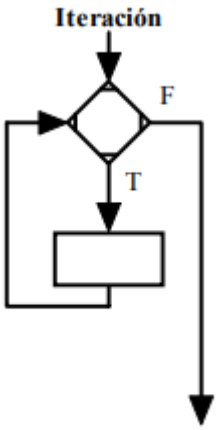
Estructuras de Control.

Tipos de estructuras de control:

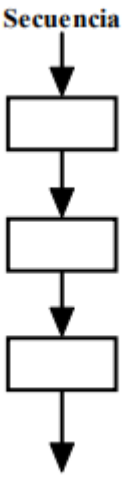
- Control condicional
  - IF-THEN
  - IF-THEN-ELSE
- Control iterativo
  - LOOP
  - EXIT-WHEN
  - WHILE-LOOP
  - FOR
- Control secuencial
  - GOTO
  - NULL



Las sentencias IF, nos permiten ejecutar una secuencia de acciones de forma condicional, es decir, el hecho de que se ejecute o no la acción, depende del valor de la condición. Hay tres variedades de sentencias IF: IF-THEN, IF-THEN-ELSE, y IF-THEN-ELSIF.



Las sentencias de tipo LOOP, permiten ejecutar una secuencia de sentencias múltiples veces. Existen tres variedades de sentencias LOOP: LOOP, WHILE-LOOP, y FOR-LOOP.



Al contrario que las sentencias IF y LOOP, las instrucciones GOTO y NULL (que son las asociadas al control secuencial), no son cruciales ni imprescindibles dentro de la programación en PL/SQL.

1. **Control condicional.**
   - **IF-THEN**

```
34  IF <condicion> THEN
35     --secuencia de sentencias;
36  END IF;
```

La secuencia de sentencias se ejecuta tan sólo si la condición es TRUE. Se pueden escribir las sentencias IF de forma completa en una sola línea.

   - **IF-THEN-ELSE**

```
40  IF <condicion> THEN
41  -- secuencia_1;
42  ELSE
43  -- secuencia_2;
44  END IF;
```

Si la condición es TRUE, se ejecutará la secuencia de instrucciones 1, en caso contrario se ejecutará la 2. Las clausulas THEN y ELSE pueden incluir sentencias IF, es decir, podemos agrupar sentencias de tipo IF.

2. **Control iterativo.**
   - **LOOP**

```
48  LOOP
49     --secuencia_de_sentencias;
50  END LOOP;
```

Se trata de la variedad más simple de la sentencia LOOP, y se corresponde con el bucle básico (o infinito), el cual incluye una secuencia de sentencias entre las palabras claves LOOP y END LOOP.

En cada iteración del bucle, se ejecutan todas las sentencias de forma secuencial. Evidentemente es raro que deseemos tener un bucle infinito en un programa, por tanto existe una manera de forzar la salida, y es la utilización de la palabra clave EXIT. Para esta palabra también tenemos dos variedades posibles: EXIT y EXIT-WHEN.

- **EXIT-WHEN**

La sentencia EXIT-WHEN, nos va a permitir salir de un bucle de forma condicional. Cuando PL/SQL encuentra una sentencia de este tipo, la condición del WHEN será evaluada… en caso de devolver TRUE, se provocará la salida del bucle… en caso contrario, se continuará la iteración.

```
56 LOOP
57    FETCH variable INTO ...
58    EXIT WHEN variable%NOTFOUND; -- Salir si se cumple la condición
59    ...
60 END LOOP;
61 CLOSE c1;
```

- **WHILE-LOOP**

La sentencia WHILE-LOOP, asocia una condición a una secuencia de instrucciones que se encuentran entre las palabras claves LOOP y END LOOP.

```
66 WHILE condicion LOOP
67    --secuencia_de_instrucciones;
68 END LOOP;
```

- **FOR**

Los rangos de un bucle FOR pueden ser literales, variables, o expresiones, pero deben poder ser siempre evaluadas como enteros.

```
72 FOR j IN 5..15 LOOP -- Asigna los valores 5,6,7,.. a j
73    IF MOD(j,5)=0 THEN -- Solo pasan los múltiplos de 5...
74       secuencia_de_instrucciones; -- j tiene valores 5,10,15
75    END IF;
76 END LOOP;
```

3. Control secuencial.

La estructura del PL/SQL es tal, que la sentencia GOTO no es necesaria de forma obligatoria. De todas formas, en algunas ocasiones, puede estar justificado su uso para simplificar un problema.

- **GOTO**

El uso de sentencias GOTO sí que puede ser más catastrófico, ya que pueden provocar un código complejo, y no estructurado, que es difícil de entender y mantener. Por lo tanto, solo hay que emplear GOTO cuando esté fuertemente justificado. Por ejemplo, cuando se desee salir de una estructura profunda (agrupación de bucles) a una rutina de manejo de errores, entonces se podría utilizar la sentencia GOTO. Esta salta a una etiqueta de forma incondicional; la etiqueta debe ser única en su ámbito, y debe preceder a una sentencia ejecutable, o a un bloque PL/SQL. Cuando se ejecuta, la sentencia GOTO transfiere el control a la sentencia o bloque etiquetados.

```
84 BEGIN
85  ...
86  BEGIN
87    UPDATE emp SET ...
88  ...
89  END;
90  ...
91     GOTO actualizar_fila;
92  ...
93 END;
```

Algunos destinos en un salto de tipo GOTO no están permitidos. De forma específica, una sentencia GOTO no puede saltar a:

- Una sentencia IF
- Una sentencia LOOP
- Un Sub-Bloque
- Fuera de un Sub-Programa

- **NULL**

La sentencia NULL, cuando se emplea sola, sin asignarla a nada, especifica literalmente "ninguna acción". Tiene dos utilidades principalmente, una es la de clarificar el código fuente para aquellos casos en los que el programa no debe hacer nada, como por ejemplo en una sentencia IF-THEN-ELSE.

```
 98  IF contador>0 THEN
 99      -- Hacemos algo...
100     ELSE
101        NULL;
102  END IF
```

La otra es cuando queramos hacer un "debug" de alguna parte del programa, ya que podemos compilar una parte del mismo, y en la parte que no hayamos programado todavía, podemos poner un NULL de hecho debemos hacerlo, ya que sino no funcionaría.

```
106  IF num_emp>500 THEN
107   -- Parte grande que queremos probar...
108  ELSE
109     NULL;
110  END IF;
```

Cursores.

Los cursores son útiles para consultas que devuelven más de una fila. Son declarados y nombrados por el programador; se manipulan por medio de sentencias específicas en las acciones ejecutables del bloque.

Control de Cursores

- 1º. Crear un área SQL específica ⟶ **DECLARE**
- 2º. Identificar el juego activo ⟶ **OPEN**
- 3º. Cargar la fila actual en variables ⟶ **FETCH**
- 4º. Si todavía existen filas sin leer, volver a 3º.
- 5º. Si no existen más filas a leer ⟶ **CLOSE**

Declaración de un cursor.

```
113  DECLARE
114     CURSOR <nombre> IS
115        <sentencia_select>;
116  BEGIN
117     ...
118  END;
```