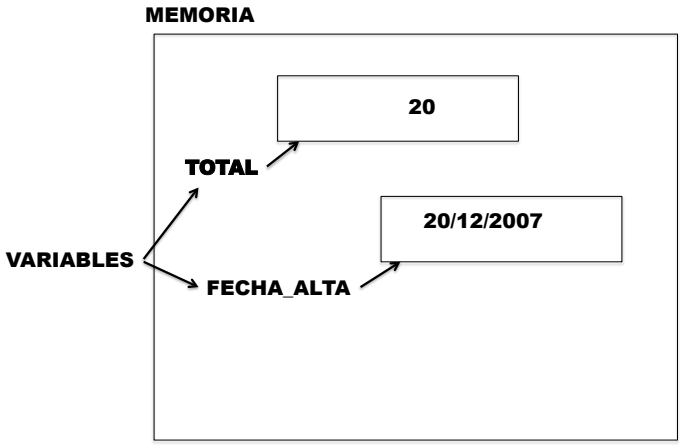
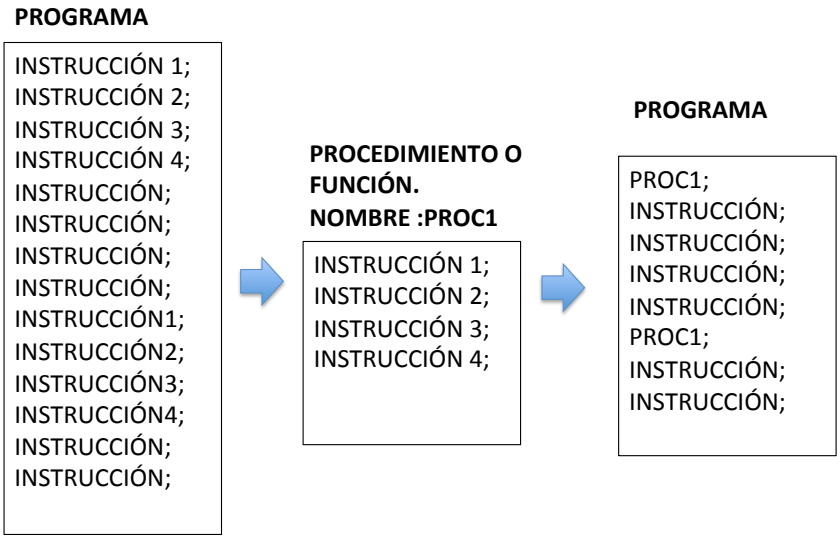
UT 6: ACLARAR CONCEPTOS DE PL/SQL

En primer lugar decir que en esta unidad se introducen conceptos de programación que se unen la lo que hemos explicado en otras unidades sobre base de datos (SELECT, INSERT, UPDATE, DELETE).
Voy a intentar aclarar algunos conceptos básicos de manera sencilla y fácil, para ver si podéis entenderlos:

- **PL/SQL** es un lenguaje que nos permite realizar un programa, es decir, realizar instrucciones o sentencias una detrás de otra que Oracle : 1º comprobará que no tengan errores de sintaxis, 2º interpretará y 3º ejecutará para darnos un resultado.
- **Variable**: todos los lenguajes de programación utilizan variables. ¿Qué es una variable?. Todos sabemos o deberíamos saber que una parte importante de los ordenadores es la memoria. Los programas para realizar determinadas operaciones deben utilizar variables para almacenar información. Una variable es un trocito de memoria que los programas van a utilizar para almacenar información de diferentes tipos (numéricos, alfanuméricos, fecha,...). Éstas variables tienen un nombre y son de un tipo determinado.



- **¿Qué es modular un programa usando procedimientos y funciones?:** para explicar este concepto lo voy a realizar gráficamente



Como vemos en la figura anterior no encontramos con un programa en el que una serie de instrucciones se repiten, lo que hemos hecho para evitar que se repitan estas instrucciones es sacarlas fuera realizando un procedimiento o función y finalmente llamar a éstos cuando se necesiten y así no tener que repetir las sentencias.

- **Procedimiento o Función :** programa PL/SQL que está almacenado en la base de datos y podremos ejecutarlo cuantas veces necesitemos.
- Vamos a explicarlo con un ejemplo: supongamos que en diferentes programas necesitamos calcular la suma de los salarios de los empleados, para no tener que repetir el proceso del cálculo realizamos un procedimiento o una función llamado SUMAR_SALARIOS.
- Posteriormente necesitamos calcular la suma de un departamento concreto. Para poder realizar la suma de cualquier departamento y que el procedimiento o función sirva, necesitaremos pasarle de alguna manera el valor del departamento al procedimiento o función para que lo calcule. Esto es lo que vamos a denominar parámetro de entrada, una variable que va usar el procedimiento o función para realizar el cálculo.
- Una vez que se ha realizado el cálculo de la suma de los salarios dentro del procedimiento, de alguna manera tenemos que pasárselo al programa. Esto se hace pasándole el valor en lo que vamos a llamar parámetros de salida.
- La diferencia que hay entre un procedimiento y una función es que la función puede devolver como salida un valor en el propio nombre de la función, no así el procedimiento.
- Vamos a realizar un procedimiento y una función con el ejemplo anterior.

```
CREATE OR REPLACE PROCEDURE PSUMA_SALARIOS(P_DEPT_NO NUMBER, P_SUMA OUT NUMBER)
IS
        SUMA NUMBER(8,2);
BEGIN
        SELECT  SUM(SALARIO)  INTO SUMA
                FROM EMPLE
                WHERE DEPT_NO=P_DEPT_NO;
        P_SUMA:=SUMA;
END;

CREATE OR REPLACE FUNCTION FSUMA_SALARIOS(P_DEPT_NO NUMBER)
RETURN NUMBER
IS
        SUMA NUMBER(8,2);
BEGIN
        SELECT  SUM(SALARIO)  INTO SUMA
                FROM EMPLE
                WHERE DEPT_NO=P_DEPT_NO;
        RETURN SUMA;
END;
```

Para poder llamar desde un bloque a un procedimiento se hará:

```
DECLARE
        VSUMA NUMBER;
BEGIN
        PSUMA_SALARIOS(10, VSUMA);
        DBMS_OUTPUT.PUT_LINE('LA SUMA DE LOS SALARIOS DEL DEPARTAMENTO 10 ES:'||VSUMA);
END ;
```

Para poder llamar desde un bloque a una función se hará:

```
DECLARE
        VSUMA NUMBER;
BEGIN
        VSUMA:=FSUMA_SALARIOS(10);
        DBMS_OUTPUT.PUT_LINE('LA SUMA DE LOS SALARIOS DEL DEPARTAMENTO 10 ES:'||VSUMA);
END ;
```

- **SELECT ...INTO**: En PL/SQL cuando ejecutamos una sentencia dentro de un bloque BEGIN...END sólo puede devolver una fila, en el caso de que no devuelva ninguna o devuelva mas de una se produce un error y salta lo que llamamos en PL/SQL una excepción.
- Las columnas que se seleccionen no se pueden usar directamente, hay que almacenarlas en variables.

```
DECLARE
        VAPELLIDO     VARCHAR2(40);
        SALARIO       NUMBER(5);
BEGIN
        SELECT APELLIDO, SALARIO INTO VAPELLIDO, VSALARIO
                FROM EMPLE
                WHERE EMP_NO=7900;
        DBMS_OUTPUT.PUT_LINE(' NOMBRE:'||VAPELLIDO||' SALARIO: '||VSALARIO);
END;
```

- Cuando necesitamos realizar un proceso sobre varios registros, como no podemos utilizar SELECT...INTO.., utilizaremos lo que denominamos **CURSORES.**

- **Pasos para utilizar un cursor:**

1. **Declarar el cursor:** Imaginemos que nos piden realizar un procedimiento que nos aumente el sueldo de todos los empleados, pero no a todos por igual: a los empleados que tengan hijos se les aumentará el salario un 10% y a los que no tengan hijos un 5%. Tengo que realizar una SELECT con todos los empleados, que normalmente devolverá más de 1, con lo cual necesitaré usar un CURSOR.
   Seleccionaré en el cursor solo las columnas que necesito para realizar los cálculos: el nº de hijos para saber el porcentaje hay que aplicar y el Nº de empleado para actualizar el salario.

```
DECLARE
        CURSOR CEMPLE IS
                SELECT EMP_NO,NRO_HIJOS FROM EMPLE;
```

2. **Abrir el cursor:**

```
        OPEN CEMPLE;
```

3. **Leer un registro** y almacenarlo en memoria:

```
        FETCH CEMPLE INTO VEMPLE;
```

   Para poder recorrer todos los registros la sentencia de lectura deberá de estar en un bucle para ir leyéndolos todos.

4. **Cerrar el cursor:**

```
        CLOSE CEMPLE;
```

   EJEMPLO COMPLETO: este procedimiento no tiene parámetros porque hay que modificar todos los empleados:

```
CREATE OR REPLACE PROCEDURE ACTUALIZAR
IS
        CURSOR CEMPLE IS
                SELECT EMP_NO,NRO_HIJOS FROM EMPLE;  -- Declarar el cursor
        VEMPLE CEMPLE%ROWTYPE; -- Declara una variable del mismo tipo que CEMPLE
                                        -- para poder almacenar cada registro
```

```
BEGIN
        OPEN CEMPLE;  -- Abrir el cursor
        LOOP  -- Bucle para recorrer todos los registros que haya devuelto la SELECT
                FETCH CEMPLE INTO VEMPLE;  -- Lee un registro y lo almacena en VEMPLE
                EXIT WHEN  -- Salir si no hay mas registros

                IF VEMPLE.NRO_HIJOS>0 THEN
                        -- Si el nro de hijos es mayor de 0 actualiza su salario increm 10%
                        UPDATE EMPLE SET SALARIO=SALARIO*1.1
                        WHERE EMP_NO=VEMPLE.EMP_NO;
                ELSE
                -- Si el nro de hijos es igual a 0 actualiza su salario increm 5%
                        UPDATE EMPLE SET SALARIO= SALARIO*1.05
                        WHERE EMP_NO=VEMPLE.EMP_NO;
                END IF;
        END LOOP;
        CLOSE CEMPLE; -- Se cierra el cursor
END;
```

- **Triggers:** en un programa PL/SQL que nosotros realizamos pero que salta (se ejecuta) automáticamente cuando realizamos un INSERT, UPDATE o DELETE en una tabla.
- En los trigger para acceder a los valores anteriores o posteriores de las columnas de las tablas y poder preguntar por los valores, tenemos que usar : NEW y : OLD:
    - INSERT: Sólo podremos acceder a los nuevos valores(:NEW) porque los anteriores no existen (:NEW.SALARIO)
    - DELETE: Sólo podremos acceder a los valores anteriores (:OLD) porque los nuenvos no existen (:OLD.SALARIO)
    - UPDATE: En una actualización podremos acceder tanto a los valores anteriores a la actualización y a los nuevos valores después de la actualización.

- **Por ejemplo:** Supongamos que se quiere que cada vez que yo elimine un registro de la tabla pedidos me salga un mensaje de que se ha eliminado el registro.
  Tendremos que realizar un trigger (yo lo voy a llamar ELIMINAR_PEDIDO) con las siguientes sentencias:

```
CREATE OR REPLACE TRIGGER ELIMINAR_PEDIDO
        BEFORE DELETE P_PEDIDOS          -- Después de eliminar en la tabla P_PEDIDOS
        FOR EACH ROW                     -- Para todos los registros que se eliminen
BEGIN -- Instrucciones que se ejecutarán : visualizar un mensaje
        DBMS_OUTPUT.PUT_LINE ('BORRADO EMPLEADO'|| '---'||OLD.EMP_NO);
END;
/
```

Tarea para BD06.

ACTIVIDAD 1

Realizaremos una función llamada **CALCULAR_PEDIDO** a la que le demos un código de pedido como entrada y nos devuelva como salida el total de ese pedido, a cada producto del pedido se le realizará un descuento sobre el precio de venta que dependerá del nº de unidades pedidas:

- Si el nº de unidades está entre 0 y 5: no habrá descuento
- Si en nº de unidades pedidas está entre 6 y 10 el descuento es del 5%
- Si el nº de unidades pedidas está entre 11 y 15 el descuento es del 7%
- Si el nº de unidades pedidas es >15 el descuento será del 10%.

El total de un producto será el resultado de multiplicar el precio de venta (con el descuento realizado) por en nº de unidades pedidas.

ACLARACIONES

- La cabecera de la función será :

```
CREATE OR REPLACE FUNCTION CALCULAR_PEDIDO (P_CODIGOPEDIDO IN NUMBER)
RETURN NUMBER AS
```

PASOS:

1. Pasaremos como entrada el código del pedido del que vamos a calcular el total
2. Un PEDIDO tiene uno o muchos registros en la tabla DETALLEPEDIDOS, cada uno de los registros de detallespedido es de un PRODUCTO pedido. Necesitaremos un CURSOR para seleccionar todos los PRODUCTOS de la tabla DETALLEPEDIDOS porque a cada producto hay que aplicarle un descuento.
3. Recorreremos el cursor mediante u bucle:
    a. Comprobaremos de cada producto las unidades pedidas y aplicaremos el descuento al precio de venta del producto.
    b. Una vez calculado el descuento, hay que acumular los totales de cada producto
4. Al final tendremos todos los totales que devolveremos como salida a la función:
   RETURN TOTAL;

ACTIVIDAD 2:

Realizaremos un **procedimiento** llamado **CALCULAR_CLIENTE** que dándole como entrada un código de cliente y un año, nos devuelva como salida dos parámetros : el total facturado por los pedidos que haya pagado y el total facturado por los pedidos que no haya pagado durante ese año. Sólo se tendrán en cuenta los pedidos que se hayan entregado ya.
Sólo tendremos en cuenta los clientes que hayan realizado más de dos pedido en ese año.
Si el cliente no existe saltaremos una excepción, visualizaremos un mensaje de que ese cliente no existe y finalizará el procedimiento devolviendo como salida -1 en ambos totales.
Si el cliente no tiene pedidos durante ese año, saltaremos una excepción y visualizaremos un mensaje de que ese cliente tiene pedidos ese año y finalizará el procedimiento devolviendo como salida -1 en ambos totales
Para ello por cada pedido realizaremos una llamada a la función CALCULAR_PEDIDO que nos calculará el total de cada pedido. Estos totales se irán acumulando en el total de los pedidos pagados o en el total de los pedidos no pagados.
Finalmente devolveremos como parámetros dichos totales.

ACLARACIONES:

- La cabecera del procedimiento será:

```
CREATE OR REPLACE PROCEDURE CALCULAR_CLIENTE(
       P_CODIGOCLIENTE     IN NUMBER,
       P_ANIO              IN NUMBER,
       P_TOTAL_SIPAGADOS   OUT NUMBER,
       P_TOTAL_NOPAGADOS   OUT NUMBER) AS
```

- P_CODIGOCLIENTE: parámetro de entrada para el código del cliente que vamos a calcular
- ANIO : parámetro de entrara donde vamos a pasar el año a buscar
- P_TOTAL_SIPAGADOS: parámetro de salida para dejar el total de los pedidos que ha pagado el cliente
- P_TOTAL_NOPAGADOS: parámetro de salida para dejar el total de los pedidos que no ha pagado el cliente

PASOS:

1. Necesitaremos un CURSOR para seleccionar todos los pedidos de un cliente, para saber el año de un pedido utilizaremos la función TO_CHAR para sacar el año. Una de las condiciones del cursor será una subconsulta que nos seleccione solo los clientes que hayan tenido más de 2 pedidos en ese año. Os aconsejo que primero probéis la consulta y después la pongáis en el cursor.
2. Buscaremos si el cliente existe en la tabla E_CLIENTES: hay varias formas de hacerlo, no es la más eficiente pero sí la más fácil: una SELECT COUN(*) ...., lo podéis mirar en los ejercicios que yo he dejado.
3. Si no existe el cliente, asignaremos TOTAL_SIPAGADOS:=-1, TOTAL_NOPAGADOS:=-1, y saltaremos una excepción (RAISE NOCLIENTE, yo la he llamado NOCLIENTE a este error).
4. Si el cliente existe: abrimos el cursor y leemos (FETCH) el primer registro del cursor.
5. Si al leer el primer registro no hay ninguno (CURSORPEDCLI%NOTFOUND), asignaremos TOTAL_SIPAGADOS:=-1, TOTAL_NOPAGADOS:=-1, y saltaremos una excepción (RAISE NOPEDIDOSCLIENTE).
6. En el caso de que el cliente exista y de que haya registros, seguiremos con un bucle para recorrer el cursor, el cual incluirá las siguientes sentencias:
    1. Llamaremos a la función CALCULAR_PEDIDO(VCUROSRPEDCLI.CODIGOPEDIDO), que nos devolverá el total de ese pedido.
    2. Si el pedido está pagado, acumularemos lo que nos devuelva la función en una variable llamada por ejemplo TOTAL_SIPAGADOS.
    3. Si el pedido no está pagado, acumularemos lo que nos devuelva la función en una variable llamada por ejemplo TOTAL_nNOPAGADOS.
    4. Volvemos a leer otro registro (otro pedido)
7. Finalmente asignaremos las variable TOTAL_SIPAGADOS al parámetro de salida P_TOTAL_SIPAGADOS y TOTAL_NOPAGADOS al parámetro de salida P_TOTAL_NOPAGADOS

ACTIVIDAD 3.

Queremos crear los siguientes disparadores:

1. Un disparador llamado **DISP_PEDIDOS** que salte al insertar o actualizar en la tabla **pedidos** para que no nos deje insertar o actualizar si se produce uno de los siguientes casos:

   - La fecha de pedido debe ser menor que la fecha esperada de entrega y la fecha de entrega.
   - El estado de un pedido sólo puede tener los valores P, E o D (Pendiente de entregar, Entregado o Devuelto)
   - La columna PedidoPagado sólo puede tener los valores : S o N (Pagado Si, Pagado No)

   Se lanzará una excepción mediante la cual el registro no se inserte y visualiza el mensaje adecuado

   ACLARACIONES:

   - La cabecera del trigger será:

```
CREATE OR REPLACE TRIGGER DISP_PEDIDOS
BEFORE INSERT OR UPDATE
ON E_PEDIDOS
FOR EACH ROW
BEGIN
```

   - Preguntaremos si la columna nueva fecha de pedido es mayor o igual que la nueva fecha de esperada, si es así saltaremos una excepción (RAISE_APPLICATION_ERROR)
   - Preguntaremos si la columna nueva fecha de pedido es mayor o igual que la nueva fecha de entrega , si es así saltaremos una excepción(RAISE_APPLICATION_ERROR)
   - Preguntaremos si columna nueva ESTADOENVIO es un valor correcto (P,E,D), si no lo es saltaremos una excepción (RAISE_APPLICATION_ERROR).
   - Preguntaremos si columna nueva PEDIDOPAGADO es un valor correcto (S,N), si no lo es saltaremos una excepción (RAISE_APPLICATION_ERROR).

2. Un disparador llamado **DISP_DETALLEPEDIDOS**, en el que al insertar o actualizar en la tabla **DETALLEPEDIDOS** controle que la cantidad pedida del producto sea menor que la cantidad en stock de dicho producto. Se lanzará una excepción mediante la cual el registro no se inserte y visualiza el mensaje adecuado.
   - La cabecera del trigger sería:

```
CREATE OR REPLACE TRIGGER DISP_DETALLEPEDIDOS
BEFORE INSERT OR UPDATE
ON E_DETALLEPEDIDOS
FOR EACH ROW
DECLARE
```

   - Primero realizaremos una SELECT para obtener las columnas CANTIDADENSTOCK se la tabla E_PRODUCTOS
   - Si la columna nueva UNIDADESPEDIDAS es mayor de la cantidad en stock saltaremos una exceptión (RAISE_APPLICATION_ERROR).