Introducción a la programación de Base de Datos

ÍNDICE

En este documento, a no ser que se indique lo contrario, se reflejan los comandos SQL referidos al Sistema Gestos de Base de Datos (SGBD) Oracle Database.

1. Introducción PL/SQL (Procedural Language / Structured Query Language)

PL/SQL es un lenguaje de programación que amplía la funcionalidad de SQL. Permite **usar sentencias SQL** para manipular datos y **sentencias de control de flujo** para procesar los datos.

Los programas creados con PL/SQL se pueden almacenar en la base de datos como cualquier otro objeto (tablas, índices, vistas…), quedando disponibles para su ejecución.

PL/SQL fue creado por Oracle y actualmente todos los gestores de bases de datos utilizan algún lenguaje parecido para poder programar las bases de datos.

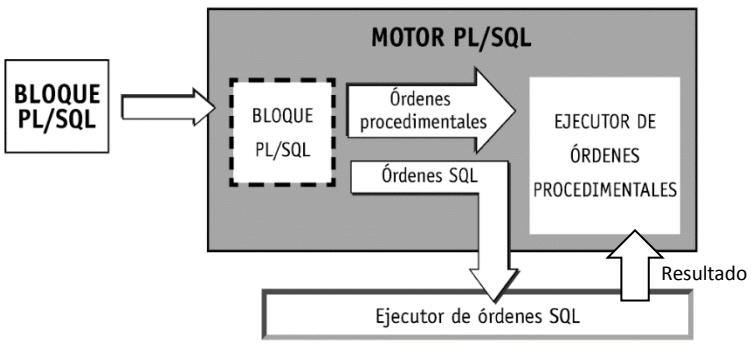
- **Principales ventajas:**
  - PL/SQL combina la potencia de SQL para la manipulación de datos, con la potencia de los lenguajes de programación para procesar los datos.
  - La ejecución se produce en el servidor, lo que implica menor tráfico en la red y menos carga en el cliente. Además el código se agrupa en bloques, lo que mejora el rendimiento.
  - Los disparadores de bases de datos permiten implementar reglas complejas de negocio y auditoría de datos. Por ejemplo, si se introduce un registro en una tabla de facturas se pueden recalcular de forma automática campos de otras tablas, o se puede invalidar la operación si no se cumplen ciertos requisitos de negocio.

- **Arquitectura**

PL/SQL es una tecnología **integrada en el servidor Oracle** capaz de ejecutar bloques de código en coordinación con el motor SQL.

También es posible que esté integrada en herramientas externas al servidor.

Antes de almacenarse el programa PL/SQL, el código es analizado sintácticamente y se comprueba que las referencias a objetos de la base de datos sea correcta (por ejemplo que las tablas existan).



En caso de que todo sea correcto, el motor PL/SQL se encarga de enviar al motor SQL los comandos SQL integrados en el bloque.

Por último ejecutará las órdenes procedimentales PL/SQL combinándolas con el resultado de los comandos SQL.

Normalmente el motor PL/SQL está en el servidor de la base de datos (SGBD), pero en caso de que esté en una herramienta externa, ejecutará las órdenes procedimentales y enviará los comandos SQL al servidor.

- **Interacción con el usuario**

  PL/SQL está diseñado para trabajar con grandes volúmenes de información eficazmente, pero no para interactuar con el usuario. **No dispone de órdenes que permitan capturar datos desde teclado ni mostrar resultados por pantalla**.

  No obstante, **con fines de depuración de programas**, Oracle incorpora el paquete DBMS_OUTPUT (permite visualizar textos en pantalla) y variables de sustitución (permite introducir valores por teclado).

- **Paquete**

  Un paquete es un contenedor que agrupa programas de la base de datos. En un entorno de producción se crean paquetes por área de negocio o para agrupar utilidades que dan soporte a otras aplicaciones.

  Para ejecutar un programa que está contenido en un paquete se utiliza la nomenclatura:

  ```
  nombrePaquete.nombrePrograma
  ```

  Algunos paquetes ya están predefinidos en la base de datos. Los más relevantes:

  - STANDARD: incorpora funciones de uso general (ROUND, NVL, ABS, TO_CHAR…).

    Este es un caso particular en el que no se indica el nombre del paquete en las llamadas a los programas. Se ejecuta `round(25.6)` en lugar de `standard.round(25.6)`.

  - DBMS_OUTPUT: incorpora funciones para depurar programas PLSQL. Para emplearlo debemos conocer dos utilidades:

`set serveroutput on|off;`	Comando que habilita/deshabilita la salida por pantalla generada por el bloque DBMS_OUTPUT.
`dbms_output.put_line('texto');`	Muestra un texto por pantalla

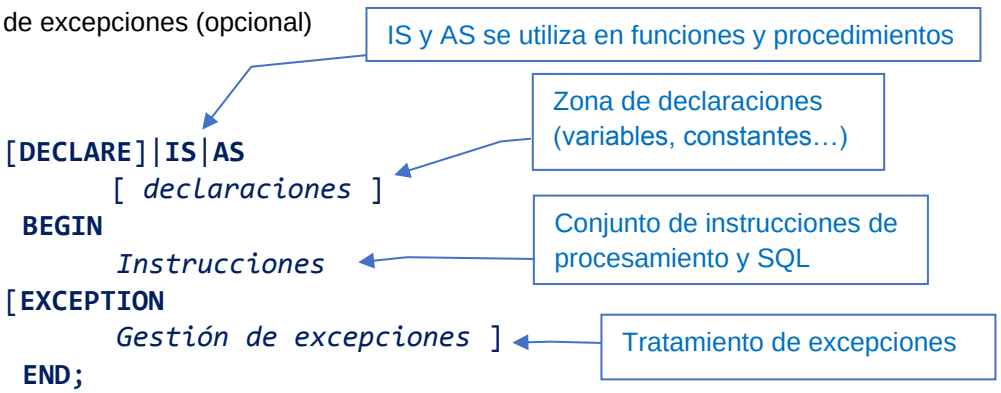
2. Conceptos básicos

2.1. Bloque

El bloque es la unidad de trabajo en PL/SQL. Se divide en tres partes claramente definidas:

- Zona de declaraciones (opcional)
- Procesamiento de instrucciones
- Tratamiento de excepciones (opcional)

Sintaxis básica:



El bloque mínimo tendrá la siguiente estructura:

```
BEGIN
    Instrucciones
END;
```

Ejecutar el primer bloque.

Tablas EMPLE y DEPART: crear un bloque que borre el departamento 20, moviendo a todos sus empleados a un nuevo departamento 99.

Paso 1) Ejecuta los comandos:

```
set autocommit off;  --Permite marcha atrás de los cambios con rollback
set serveroutput on; --Habilita la salida del bloque DBMS_OUTPUT
```

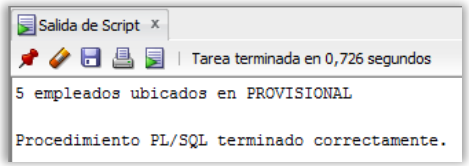
Paso 2) Escribe el bloque (disponible en el fichero bloque.sql):

```
declare
  v_num_empleados number(2);
begin
  insert into depart values (99,'PROVISIONAL',NULL);
  update emple set dept_no=99 where dept_no=20;
  v_num_empleados := SQL%ROWCOUNT;
  delete from depart where dept_no=20;
  dbms_output.put_line(v_num_empleados || ' empleados ubicados en PROVISIONAL');
exception
  when others then raise_application_error(-20000,'Error en aplicación');
end;
/
```

|| se utiliza para concatenar dos textos

Esta barra indica el fin de bloque, manda su ejecución inmediata

Ejecuta el bloque (como si fuese un comando normal).



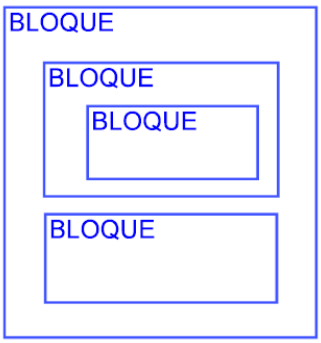
Observa que la salida del bloque se presenta en una ventana distinta “salida de script” a la que se utiliza para el resultado de la ejecución de comandos SQL “resultado de la consulta”.

Este es un bloque anónimo (sin nombre), por lo que **no se queda guardado en la base de datos.**

Paso 3) Comprueba que se han realizado los cambios en las tablas EMPLE y DEPART. Después ejecuta un rollback para volver a la situación inicial.

Es posible realizar anidación de bloques, introduciendo bloques dentro de la sección de instrucciones y de tratamiento de excepciones (no en la zona declarativa):

```
DECLARE
…
BEGIN
…
    DECLARE
    …
    BEGIN
    …
    EXCEPTION
    …
    END;
…
EXCEPTION
…
END;
```



2.2. Comentarios

```
-- Este es un comentario en línea

/*  Este es un comentario
    de varias líneas */
```

2.3. Soporte para comandos SQL

En un bloque PL/SQL se pueden ejecutar comandos SQL para manipulación y consulta de datos.

Los comandos de manipulación se integran directamente. Por ejemplo:

```
delete from emple where emp_no = 1050;
update emple set comision := 600 where emp_no = 1500;
insert into emple (emp_no,apellido) values (6000,'LARA');
```

Los comandos de consulta de datos no muestran el resultado en pantalla directamente, sino que lo guardan en un área de memoria denominada cursor, a la que se accede utilizando variables. Por esa razón la integración de un comando SELECT dentro de un bloque PL/SQL no es directa. Por ejemplo:

```
select apellido from emple where emp_no = 1050; --Dará error
```

2.4. Operadores

Aritméticos	+	Suma
	-	Resta
	*	Multiplicación
	/	División
Comparación	=	Igual a
	>	Mayor que
	>=	Mayor o igual que
	<	Menor que
	<=	Menos o igual que
	!= <>	Distinto de
	is [not] null	[no] tiene valor 'null'
	like	Igual a (se emplea en textos. Caracteres especiales: _ %)
Conjuntos de valores	in	Comprueba si un valor pertenece a un conjunto de valores (lista)
	Between and	Comprueba si un valor está en un rango de valores
Lógicos	and	Devuelve TRUE si las dos condiciones son verdaderas
	or	Devuelve TRUE si al menos una de las dos condiciones es verdadera
	not	Devuelve TRUE si la condición es falsa y FALSE si es verdadera
Concatenación	||	Concatena dos textos
Asignación	:=	Asigna un valor a una variable

2.5. Variables

Las variables del programa se definen en la **zona de declaraciones** del bloque. La sintaxis es:

```
nombreVariable tipoDato;
```

Se puede inicializar una variable al declararla. Ejemplos:

```
v_numero number(2) default 6;
v_numero number(2) := 6;
```

- **Tipos de datos**

Los tipos de datos que se pueden emplear son:

TIPO DE DATOS	EJEMPLO
Los mismos que se emplean en los datos de las tablas (varchar2, number, date…)	`precio number(5,2);`
Algunos tipos propios (boolean)	`repetidor boolean;`
El mismo tipo que un campo de una tabla existente: *tabla.campo*%type	`sueldoMedio emple.salario%type;`
El mismo tipo que un registro de una tabla existente: *tabla*%rowtype	`empleado emple%rowtype;`

Nota: en este documento será habitual definir los nombres de las variables con el prefijo v_ para distinguirlas de campos de tablas de la base de datos (por ejemplo v_precio…). Esto no es obligatorio.

- **Variables de sustitución**

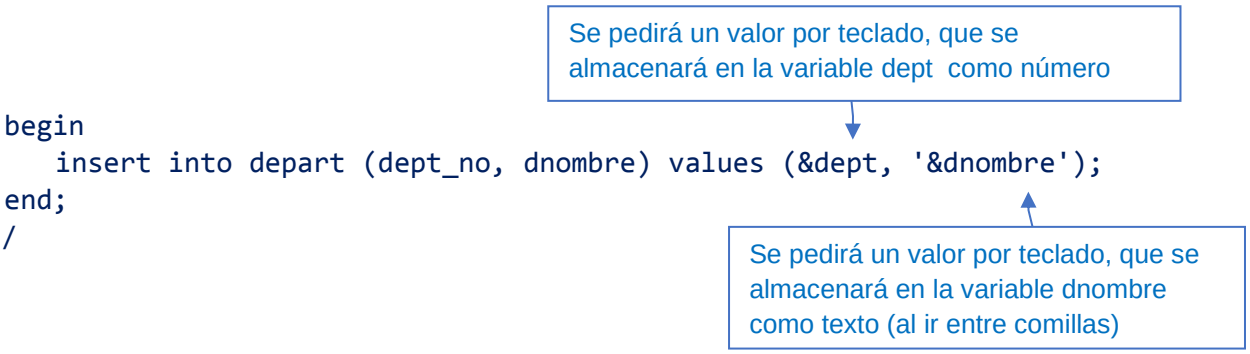
Con la finalidad de depurar programas (nunca con la de que sean empleadas por usuarios finales), es posible tomar valores por teclado a través de variables de sustitución.

Solo pueden emplearse en bloques PL/SQL anónimos, y fuera de estructuras repetitivas (bucles LOOP).

Estas variables no es necesario declararlas en la zona de declaración, sino que se utilizan directamente poniendo su nombre precedido por &:

```
 &nombreVariable  --se guardará un valor numérico
'&nombreVariable' --se guardará un valor de tipo texto
```

Prueba a ejecutar el siguiente bloque:




```
begin
   insert into depart (dept_no, dnombre) values (&dept, '&dnombre');
end;
/
```

Si se quiere utilizar una variable de sustitución varias veces en el programa, será necesario cargar su valor en una variable normal. Por ejemplo:

```
declare
    v_dept number(2);
    v_dnombre varchar2(14);
begin
    v_dept := &dept;
    v_dnombre := '&dnombre';
    insert into depart (dept_no, dnombre) values (v_dept, v_dnombre);
    dbms_output.put_line('Se ha insertado: '||v_dept||' '||v_dnombre);
end;
 /
```

Actividad. Variable de sustitución

Crea un bloque de programación que, a través de una variable de sustitución, lea el nombre del usuario y le salude con la frase: “Hola ….”.

2.6. Cursor implícito

Un cursor implícito es un área de memoria donde se guarda información sobre el último comando SQL ejecutado en un bloque PL/SQL.

Un cursor implícito no es necesario declararlo como cursor en la zona de declaraciones.

Tiene limitaciones:

- Los datos del cursor estarán **disponibles únicamente tras ejecutar el comando sql**, no podrán recuperarse más adelante.
- **No es válido para consultas SELECT que devuelvan más de un registro**

Para salvar estas limitaciones se utilizan los cursores explícitos.

▪ Cursor implícito para comandos LMD

Un cursor implícito dispone de varios atributos con información sobre la ejecución del último comando, entre ellos:

➡ SQL%ROWCOUNT devuelve el número de filas afectadas por el último comando (INSERT, UPDATE, DELETE).

Ejemplo. Borrar los empleados de un determinado oficio.

```
begin
  delete from emple where oficio='&oficio';
  dbms_output.put_line('Se han borrado '||sql%rowcount ||' empleados');
end;
/
```

Actividad. Cursor implícito en comandos LMD

> Tabla EMPLE: Crea un bloque que aumente en 100 euros la comisión de los empleados que tienen comisión (no es null) y ésta es inferior a 1000 euros.
> Muestra un mensaje informativo que indique a cuántos empleados se les ha actualizado la comisión.

- **Cursor implícito para recuperar datos de una consulta**

En el caso de los datos devueltos por una consulta **un cursor implícito soportará solo los SELECT que tienen un único registro como resultado** (solo devuelve una fila).

Si se utiliza un cursor implícito para obtener datos de una consulta que devuelve más de una fila, se producirá la excepción TOO_MANY_ROWS. Si no devuelve ninguna fila se generará NO_DATA_FOUND.

Si se quiere recuperar el registro resultado de una consulta, solo será necesario declarar las variables que van a recoger los datos devueltos. El tipo de datos de esas variables debe ser el mismo de los datos devueltos por la consulta.

El formato básico es:

Ejemplo. Visualizar el apellido y salario del empleado con número de empleado 7698 (tabla EMPLE).

```
declare
  v_ape varchar2(30);
  v_sal number(7);
begin
  select apellido, salario into v_ape, v_sal from emple where emp_no=7698;
  dbms_output.put_line(v_ape||' '||v_sal);
end;
/
```

Actividad. Cursor implícito en consultas

a) Modifica el bloque del ejemplo para definir las variables empleando %type.

b) Modifica el bloque del ejemplo para que también muestre el nombre del departamento en el que trabaja el empleado (campo dnombre).

c) Modifica el bloque anterior para que, en lugar de buscar los datos del empleado 7698, se pida a través de una variable de sustitución el número de empleado.

d) Crea un bloque que muestre el oficio del empleado con más salario. Lo que se mostrará en pantalla será el texto: `El oficio mejor pagado es … con un salario de …`

e) Crea un bloque que muestre todos los datos del empleado 7876. Define una única variable con %rowtype.

2.7. Gestión de excepciones: excepciones predefinidas

```
EXCEPTION
    WHEN … THEN
       Instrucciones
    WHEN … THEN
       Instrucciones
   [WHEN OTHERS THEN
       Instrucciones ]
```

Las excepciones sirven para tratar errores, situaciones y mensajes de aviso. En el bloque PL/SQL existe una zona de excepciones cuyo formato es:

Una excepción está asociada a un determinado evento. Cuando ocurre ese evento, automáticamente se deja de ejecutar la sección de instrucciones del bloque PL/SQL y se pasa a la sección de excepciones.
En ella se busca el manejador (WHEN) del evento que se ha producido y ejecuta el código asociado. Si no lo encuentra se ejecutará el manejador genérico (WHEN OTHERS).

El manejador se encargará de realizar una acción, por ejemplo dar marcha atrás a los cambios (rollback) o mostrar un mensaje de error o aviso.

Una vez ejecutado el código asociado a la excepción, se finaliza el programa PL/SQL.

Es posible crear excepciones personalizadas, aunque existen excepciones predefinidas como:

- NO_DATA_FOUND: un comando SELECT no ha devuelto ninguna fila (no tiene por qué ser un error)
- TOO_MANY_ROWS: un comando SELECT ha devuelto más de una fila (requiere usar un cursor explícito)

Ejemplo. Visualizar el apellido y salario del empleado con un determinado oficio programando las excepciones posibles.

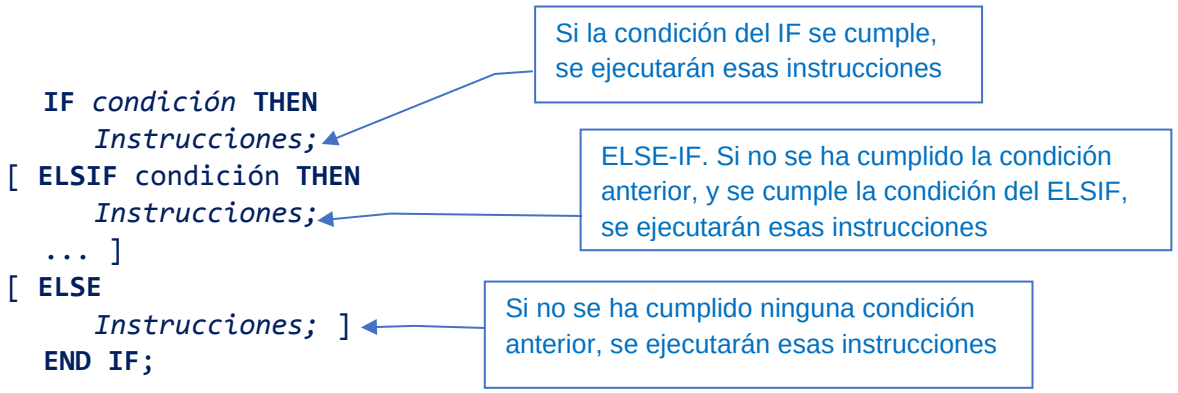
```
declare
  v_ape emple.apellido%type;
  v_sal number(7);
  v_oficio emple.oficio%type;
begin
  v_oficio := '&oficio';
  select apellido, salario into v_ape, v_sal from emple where
   oficio=v_oficio;
  dbms_output.put_line(v_ape||' '||v_sal);
exception
    when NO_DATA_FOUND then
        dbms_output.put_line('No hay datos para el oficio '||v_oficio);
    when TOO_MANY_ROWS then
        dbms_output.put_line('Se han encontrado varios registros para '||v_oficio);
    when OTHERS then
        dbms_output.put_line('Ha habido algún error');
end;
/
```

- Prueba a introducir, en la ejecución, los oficios PRESIDENTE, CONSULTOR y ANALISTA.
- Comprueba la diferencia en cada caso entre definir las excepciones y no hacerlo (borrar la sección de excepciones del bloque).

2.8. Estructuras de control

Muchas estructuras de control requieren evaluar una condición que puede tener tres resultados: true, false o null. Se considerará que se cumple la condición solo si el resultado es true.

- **If else**



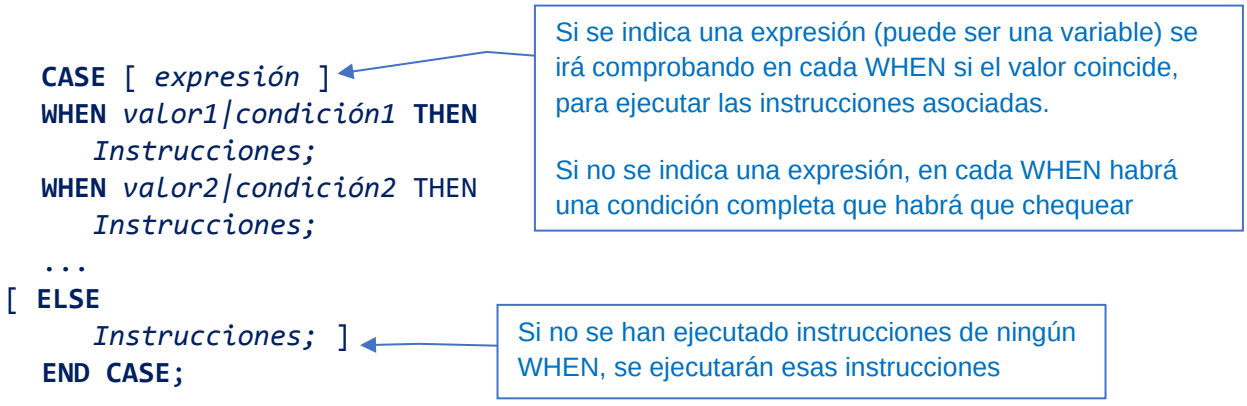
Ejemplos:

```
IF v_edad >= 18 THEN
  v_texto := 'adulto';
END IF;
```

```
IF v_edad >= 18 THEN
  v_texto := 'adulto';
ELSE
   v_texto := 'menor';
END IF;
```

```
IF v_edad >= 18 THEN
  v_texto := 'adulto';
ELSIF v_edad > 12 THEN
   v_texto := 'adolescente';
ELSE
   v_texto := 'niño';
END IF;
```

- **Case**



Ejemplos:

```
CASE v_opcion
WHEN 1 THEN
  v_texto := 'primera';
WHEN 2 THEN
  v_texto := 'segunda';
ELSE
  v_texto := 'otra';
END CASE;
```

```
CASE
WHEN v_opcion = 1 THEN
  v_texto := 'primera';
WHEN v_opcion = 2 THEN
  v_texto := 'segunda';
ELSE
  v_texto := 'otra';
END CASE;
```

Puede utilizarse IF-ELSIF-ELSE o CASE de forma equivalente:

```
IF v_edad >= 18 THEN
  v_texto := 'adulto';
ELSIF v_edad > 12 THEN
   v_texto := 'adolescente';
ELSE
   v_texto := 'niño';
END IF;
```

```
CASE
WHEN v_edad >= 18 THEN
  v_texto := 'adulto';
WHEN v_edad >= 12 THEN
  v_texto := 'adolescente';
ELSE
  v_texto := 'niño';
END CASE;
```

Actividad. Estructuras de control condicionales

Se desea crea un bloque de programación que aumente el salario de un empleado en función del personal que tiene a su cargo (se realizará un UPDATE de la tabla EMPLE):

- Se introducirá el número de empleado al que subir el sueldo a través de una variable de sustitución.
- La subida será de: 0 euros (no tiene empleados a su cargo)
  80 euros (tiene 1 empleado a su cargo)
  100 euros (tiene 2 empleados a su cargo)
  150 euros (tiene 3 o más empleados a su cargo)
- Aparecerá un mensaje con el formato: “Al empleado … se le ha realizado una subida salarial de … euros, al tener … empleados a su cargo.”

a) Utiliza una estructura IF-ELSIF-ELSE
b) Utiliza una estructura CASE con expresión
c) Utiliza una estructura CASE sin expresión

- **Loop - while**

```
LOOP
 Instrucciones;
 EXIT [ WHEN condición ];
 Instrucciones;
END LOOP;
```

Se repetirá indefinidamente el bucle a no ser que se encuentre una instrucción EXIT sin condición, o con una condición que se cumpla (es una condición de salida: si se cumple se sale del bucle).

Las instrucciones antes de EXIT se ejecutarán como mínimo una vez.

```
WHILE condición LOOP
 Instrucciones;
END LOOP;
```

Mientras se cumpla la condición, se ejecutan las instrucciones del bucle (es una condición de continuación: si se cumple se continúa en el bucle)

Es posible que las instrucciones no se ejecuten nunca.

Ejemplos: ¿qué se visualizará en cada caso?

```
v_num := 0;
LOOP
   dbms_output.put_line(v_num);
   v_num := v_num + 1;
   EXIT WHEN v_num = 3;
   dbms_output.put_line('-');
END LOOP;
```

```
v_num := 0;
WHILE v_num < 3 LOOP
   dbms_output.put_line(v_num);
   v_num := v_num + 1;
   dbms_output.put_line('-');
END LOOP;
```

- **For**

"Índice" es una variable (no es necesario declararla) que tomará el valor "inicio" y se incrementará en 1 hasta alcanzar el valor "final".

Si se incluye REVERSE "índice" tomará el valor final y se decrementará en 1 hasta alcanzar el valor "inicio".

```
FOR índice IN [REVERSE] inicio..final LOOP
 Instrucciones;
END LOOP;
```

Ejemplos:

```
/*Cuenta del 1 al 10*/
FOR I IN 1..10 LOOP
   dbms_output.put_line(i);
END LOOP;
```

```
/*Cuenta del 10 al 1*/
FOR I IN REVERSE 1..10 LOOP
   dbms_output.put_line(i);
END LOOP;
```

Actividad. Bloque revisión salarial

Crea un bloque PL/SQL que realice la revisión salarial de un empleado (tabla EMPLE). Se introducirá como variable de sustitución el apellido de un empleado. Si ese empleado cobra por debajo de la media de su departamento, se le incrementará el sueldo un 10% (en caso contrario se dejará el sueldo sin actualizar).

Se realizará la siguiente gestión de errores: si el empleado no existe se mostrará el mensaje "El empleado … no existe". Si hay más de un empleado con ese apellido se mostrará "Hay varios empleados con el apellido …".

Para mejorar el bloque añade lo siguiente:

- Utiliza la función UPPER() para que, aunque se introduzca el apellido en minúsculas, se traduzca siempre a mayúsculas.
- Inserta mensajes informativos de la acción que ha realizado el programa. Ejemplos:

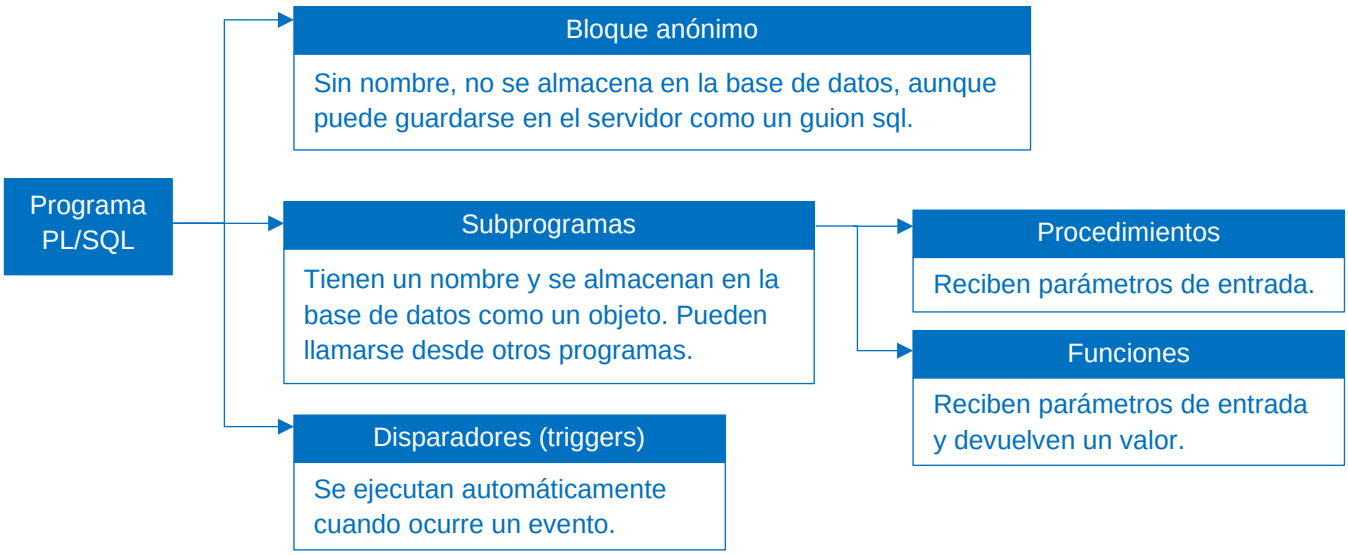
```
Se ha actualizado el salario de ARROYO de 1600 a 1760
```

```
No se ha actualizado el salario de REY
```

Nota: hay una forma de poder manipular en un programa PL/SQL el resultado de un SELECT que devuelve más de un registro, sin emplear un cursor explícito, empleando un FOR. Ejemplo:

```
begin
    for i in (select * from emple) loop
          dbms_output.put_line('Empleado '||i.emp_no||' '||i.apellido);
    end loop;
end;
```

3. Tipos de programas



Ejemplos.

Vamos a probar cada uno de los tipos de programas, aunque no los hayamos estudiado. Realiza los siguientes pasos:

1. Para que el usuario no administrador "alumno" sea capaz de crear programas que se almacenen en la base de datos, es necesario darle permisos. Con el usuario administrador ejecuta:

```
alter session set container=xepdb1;
grant create procedure to alumno; --da permiso para funciones y procedimientos
grant create trigger to alumno;
```

2. Ejecuta el bloque anónimo del fichero "tipoPrograma_bloqueAnonimo.sql". Recuerda que debes activar la salida por pantalla del bloque DBMS_OUTPUT (`set serveroutput on`).
   Este bloque no se queda almacenado en la base de datos.

3. Crea la función "cuenta_constraints" que se encuentra en el fichero "tipoPrograma_funcion.sql". Una vez creada, comprueba que aparece como objeto en el desplegable de sqlDeveloper.

   Es posible llamarla desde otros programas o comandos sql. Haz una prueba ejecutando:

```
select cuenta_constraints('EMPLE','R') from dual;
```

   Nota: "dual" es una tabla que se utiliza para hacer pruebas en Oracle.

4. Crea el procedimiento "info_constraints_tabla" que se encuentra en el fichero tipoPrograma_procedimiento.sql".

   Una vez creado, comprueba que aparece como objeto en el desplegable de sqlDeveloper.
   Observa que este procedimiento hace llamadas a la función creada antes.

   Este procedimiento puede ser llamado desde otros programas, y también puede ejecutarse directamente con el comando EXECUTE. Haz una prueba ejecutando:

```
execute info_constraints_tabla('EMPLE');
```

5. Crea el disparador "prueba_trigger" que se encuentra en el fichero tipoPrograma_disparador.sql".

   Una vez creado, comprueba que aparece como objeto en el desplegable de sqlDeveloper.

   Observa que este disparador se ejecutará automáticamente cuando se realice un INSERT sobre la tabla EMPLE.

   El código del disparador es de prueba, solamente imprime un mensaje en pantalla, aunque podría hacer cualquier acción en la base de datos.

   Para comprobar que su ejecución es automática cuando ocurre el evento, inserta el siguiente empleado:

```
insert into emple (emp_no, apellido, dept_no) values (5555,'MATEO',20);
```

Actividad. Introducción a las funciones

Aunque no hemos estudiado las funciones, basándote en el ejemplo, crea una función de nombre "media_dep" al que se le pase el número de departamento de un empleado (tabla EMPLE) y devuelva el salario medio de ese departamento, redondeado sin decimales.

Si el departamento no existe, devolverá "null".

Nota: puedes utilizar la tabla dual para probar el funcionamiento de la función.

Actividad. Introducción a los procedimientos

Aunque no hemos estudiado los procedimientos, basándote en el ejemplo, crea un procedimiento de nombre "revision_salarial" al que se le pase el apellido de un empleado (tabla EMPLE). Si ese empleado cobra por debajo de la media de su departamento, se le incrementará el sueldo un 10% (en caso contrario se dejará el sueldo sin actualizar).

Si el apellido se introduce con minúsculas el programa automáticamente lo convertirá a mayúsculas.

Se realizará la siguiente gestión de errores: si el empleado no existe se mostrará el mensaje "El empleado … no existe". Si hay más de un empleado con ese apellido se mostrará "Hay varios empleados con el apellido …".

El procedimiento debe llamar a la función "media_dep" creada anteriormente.

Para probar el procedimiento puedes emplear la instrucción EXECUTE.

Actividad. Introducción a los disparadores

Aunque no hemos estudiado los disparadores, basándote en el ejemplo, crea un trigger de nombre "check_salario" que se ejecute cuando se inserte un nuevo empleado en la tabla EMPLE.

Lo que hará será revisar el salario de ese nuevo empleado que ha insertado. Si es menor que la media de su departamento, aparecerá el mensaje informativo: "AVISO: Este empleado cobra por debajo de la media de su departamento". En caso contrario, no habrá ningún mensaje.

El disparador debe llamar a la función "media_dep" creada anteriormente.

Para probar el disparador debes insertar registros en la tabla EMPLE.