**FUNDAMENTOS DE PROGRAMACIÓN EN BASES DE DATOS (PL/SQL)**

### **1. Introducción a PL/SQL**

PL/SQL es el lenguaje de programación utilizado en Oracle, combinando comandos SQL con estructuras procedimentales para gestionar bases de datos de manera eficiente.

### **2. Elementos Léxicos de PL/SQL**

#### **Delimitadores:**

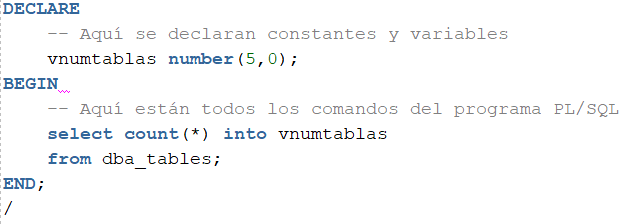
* **Asignación:** **:=**
* **Igualdad:** **=**
* **Operadores relacionales: <, >, <=, >=**
* **Diferencia: <>** o **!=**
* **Operadores aritméticos:** **+, -, \*, /**
* **Potencia:** **\*\***
* **Comentarios:** **--** (una línea), **/\* ... \*/** (varias líneas)

#### **Otros elementos:**

* **Identificadores:** Conjunto de identificadores que usa Oracle para su propia configuración.
* **Palabras reservadas:** Son las que usa el propio lenguaje PL/SQL.
* **Literales:** Valores numéricos o de texto entre comillas simples, concatenados con **||.**

### **3. Bloques de un Programa PL/SQL**

1. **DECLARE** (Opcional): Se declaran variables, constantes, cursores y excepciones definidas por el usuario.
2. **BEGIN** (Obligatorio): Contiene los comandos SQL junto a las sentencias de control PL.
3. **EXCEPTION** (Opcional): Define el manejo de errores que puedan ocurrir durante la ejecución del bloque **BEGIN**.
4. **END** (Obligatorio): Finaliza con un punto y coma.



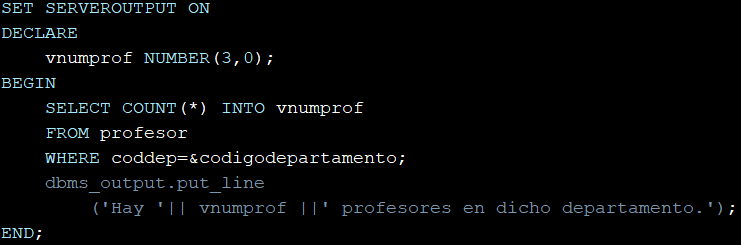
### **4. VARIABLES EN PL/SQL**

Las variables almacenan valores en memoria y pueden cambiar durante la ejecución.

* Declaración básica:



*Ejemplo: obtener el número de profesores cuyo código de departamento sea igual a uno que se pide por teclado.* **SET SERVEROUTPUT ON** *y* **dbms\_output.put\_line** *se utilizan para mostrar por pantalla el resultado.*

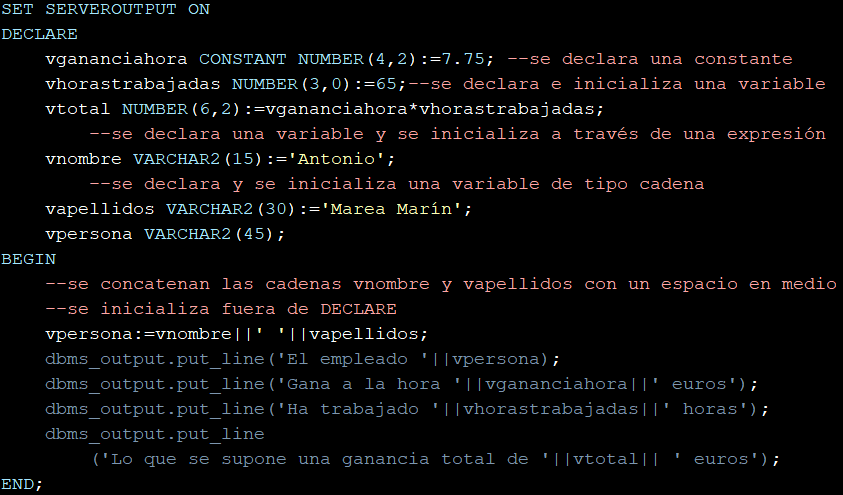


**······ Variables de acoplamiento ········**

Una vez indicado el nombre y tipo de variable, puede ser inicializada utilizando los caracteres **:=.**

Una variable **NOT NULL** siempre debe ser inicializada

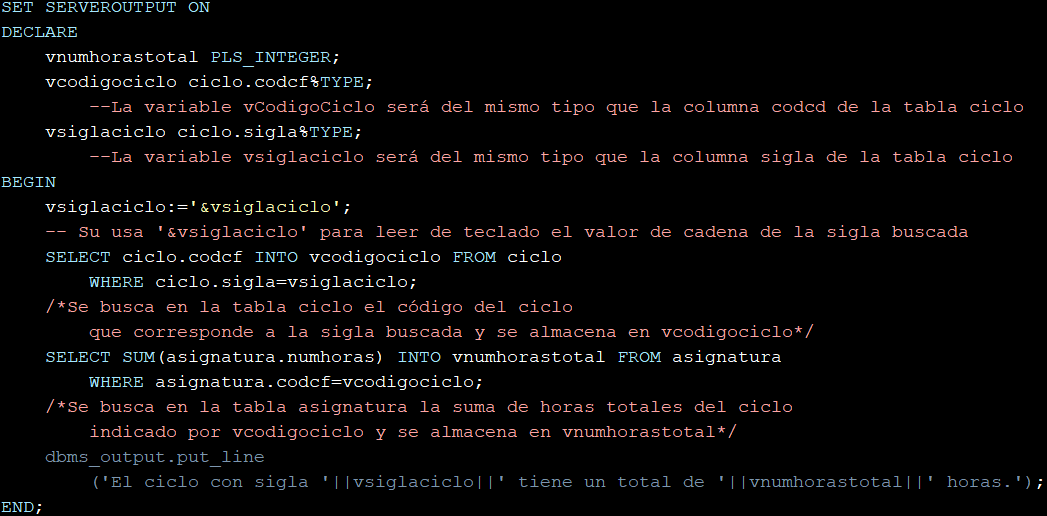
Si el valor para una variable es constante no puede variar a lo largo del programa. Se usa para ello la opción **CONSTANT**. *Como en el ejemplo:*



### **5. TIPOS DE DATOS Y ATRIBUTO %TYPE**

* El atributo **%TYPE** permite obtener el tipo de una columna o de una variable definida con anterioridad
* **SELECT INTO** almacena el resultado de una consulta en una variable

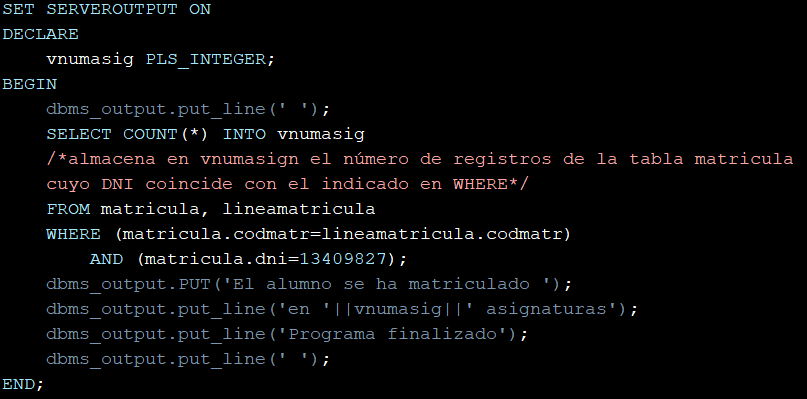
*Ejemplo: se usa el atributo %TYPE para obtener el tipo de datos de algunas columnas y la expresión SELECT INTO para almacenar el resultado de una consulta en una variable:*



### **6. ENTRADA Y SALIDA DE DATOS**

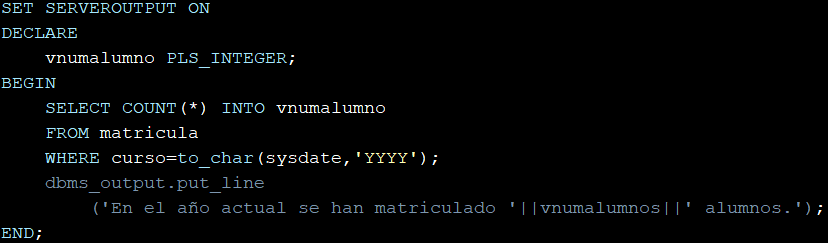
PL/SQL no está diseñado para la entrada de datos por teclado ni para la salida directa en pantalla, sino para la evaluación de datos. Para mostrar resultados se usa **DBMS\_OUTPUT.PUT\_LINE**. No permite operaciones aritméticas y siempre inserta un intro al final, evitando esto con **PUT** seguido de **PUT\_LINE**. Por defecto, el servidor no muestra la salida, por lo que se debe activar con **SET SERVEROUTPUT ON**.

*Por ejemplo, mostrar el número de asignaturas en las que se ha matriculado un alumno:*



### **7. ASIGNACIÓN A TRAVÉS DE SELECT INTO**

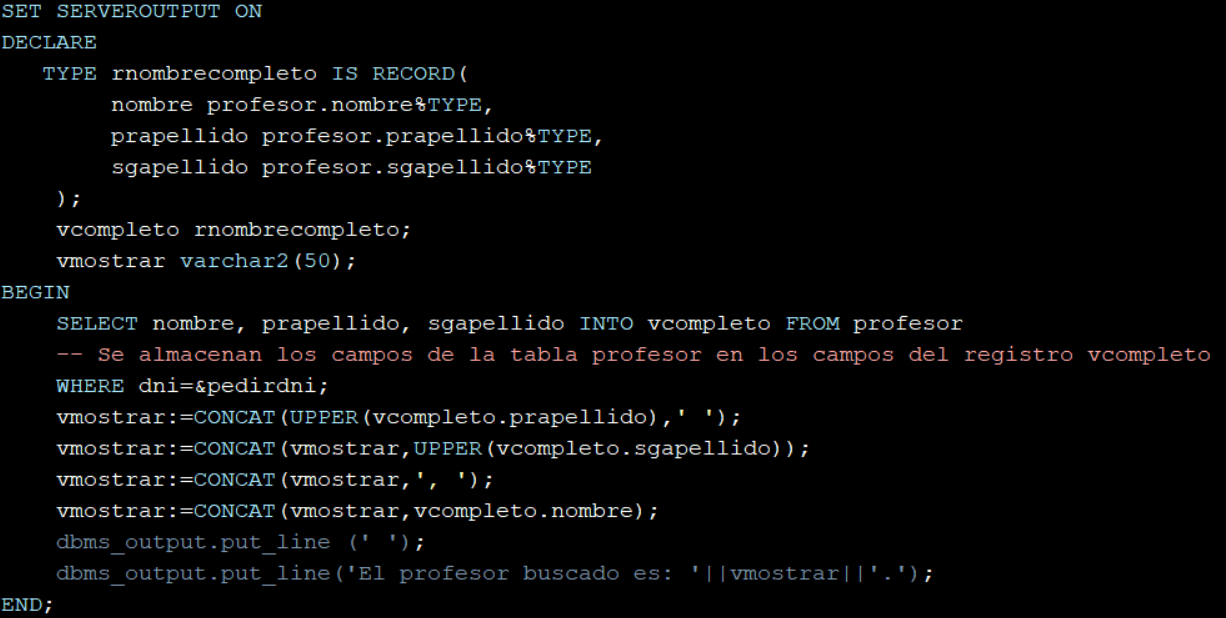
Si una consulta incluye una función de agregación como **COUNT, SUM, AVG, MIN o MAX** sin **GROUP BY**, devolverá un único valor. También ocurre si se proyecta una sola columna y la condición en **WHERE** coincide con un único registro. En estos casos, la variable usada después de **INTO** debe ser de un tipo adecuado para almacenar el resultado. *Por ejemplo, para contar los alumnos matriculados en el curso actual.*



* **PSL\_INTEGER**: tipo de datos entero en PL/SQL Oracle.
* **COUNT (\*):** devuelve el número de filas de la tabla matricula involucradas en la pregunta de WHERE
* **TO\_CHAR:** convierte la fecha/hora indicada entre paréntesis a una cadena según el formato indicado “YYYY” -solo el año con cuatro cifras-
* **SYSDATE:** devuelve la fecha del sistema

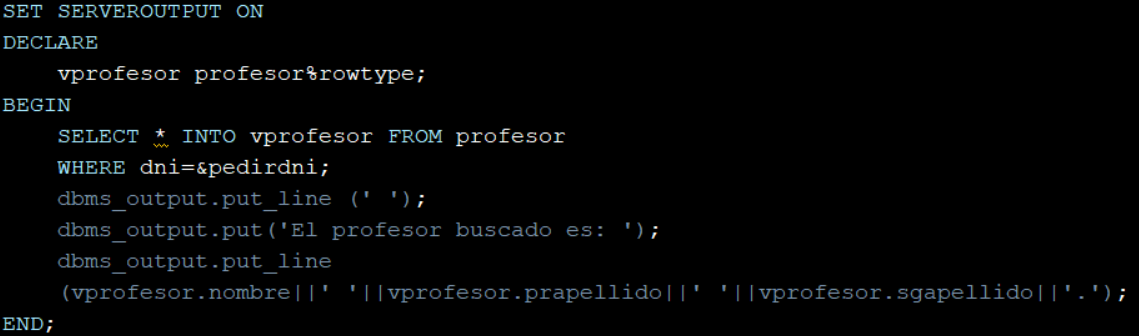
#### **7.1. La consulta devuelve una fila con varios campos.**

Cuando una consulta devuelve siempre una única fila, podemos utilizar un **RÉCORD** para almacenar su resultado. Supongamos que deseamos almacenar el nombre, primer apellido y segundo apellido resultante de una consulta por DNI y mostrarlos con el formato, PAPELLIDO SAPELLIDO, Nombre:



#### **7.2. La consulta devuelve una fila completa**

Se puede utilizar un registro completo con todas las columnas de la tabla para facilitar su gestión. Esto se logra con **%ROWTYPE**, que define los tipos de todas las columnas. En las consultas, se deben proyectar todas las columnas con **\*.**

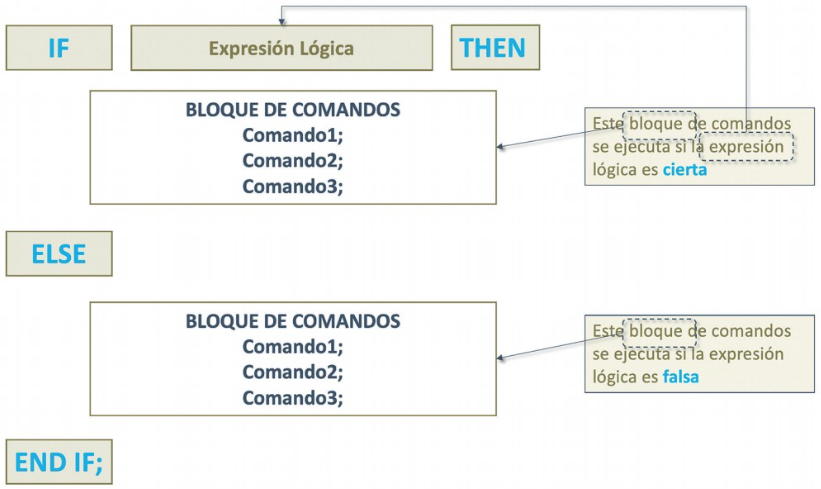


### **8. ESTRUCTURAS DE CONTROL**

Las estructuras de control permiten gestionar la secuencialidad de la ejecución de los comandos del cuerpo de un programa.

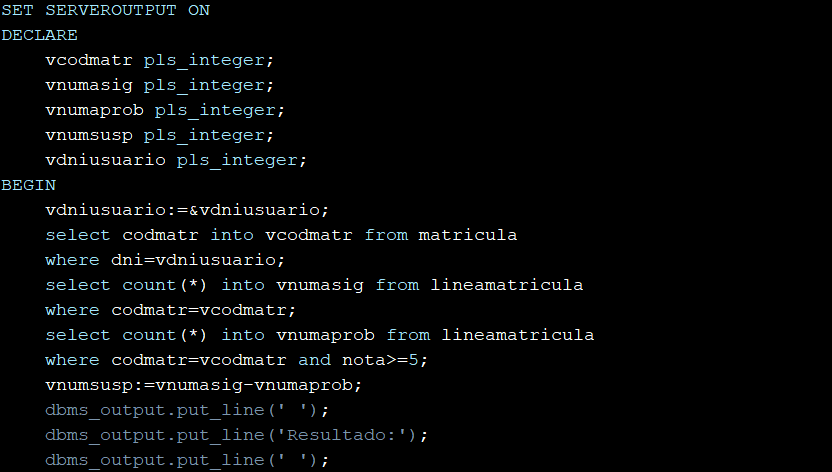
#### **8.1. Las estructuras selectivas con IF**

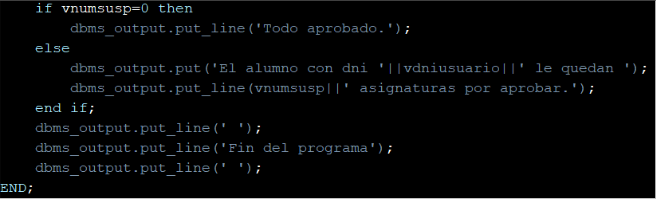
Permiten ejecutar un bloque de comandos atendiendo a si se cumple una condición o no.



En caso de que la expresión lógica sea cierta, se ejecutan los comandos incluidos en **THEN**. En caso de que sea falsa, los incluidos en **ELSE**. Una vez procesados los comandos, la ejecución salta al siguiente comando después de la etiqueta **END IF;**

*Ejemplo: Cuántas asignaturas se ha matriculado el alumno con el DNI indicado y en cuántas de ellas ha obtenido un 5 o más. Si ambos números son iguales, mostrará el texto “Todo aprobado” y, si no, “Le quedan n asignaturas por aprobar”:*





#### 

#### 

#### 

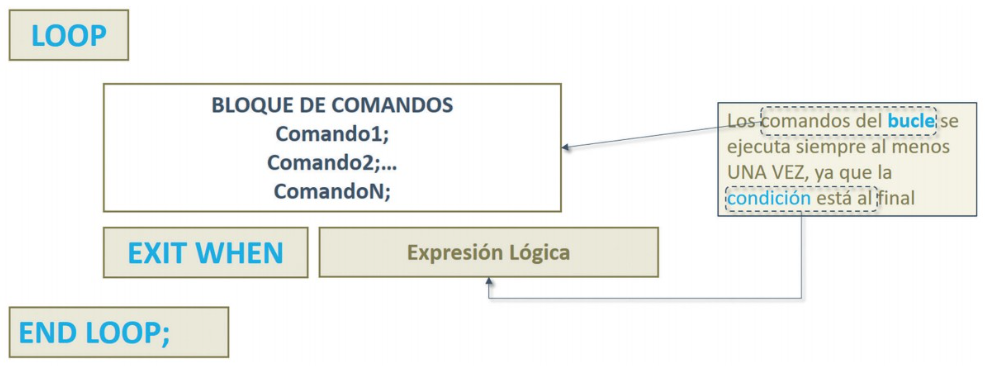
#### 

#### 

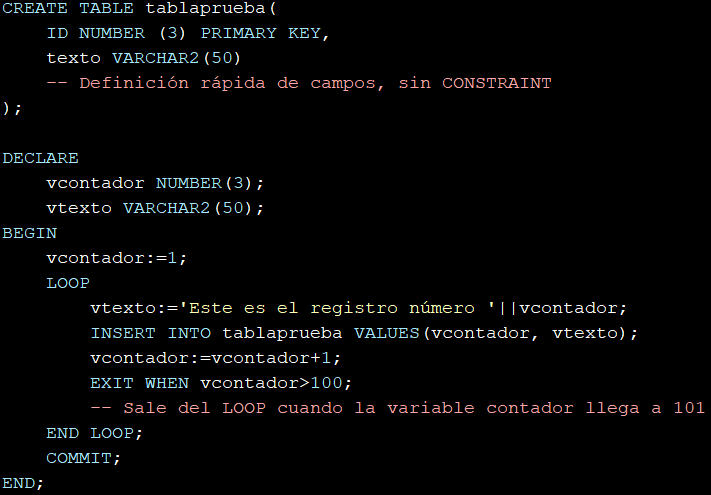
#### 

#### **8.2. Bucles (LOOP)**

Las estructuras repetitivas van a permitir ejecutar un bloque de código un número de veces. Cada una de las vueltas se denomina iteración.

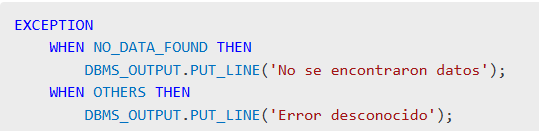


*Vamos a crear una tabla llamada tablaprueba con dos campos, id y texto. Inserta 100 registros con el texto ‘Este es el registro número N, donde N es un número que va desde 1 hasta 100. Creamos el usuario TABLAPRUEBA con contraseña tablaprueba para el ejemplo:*



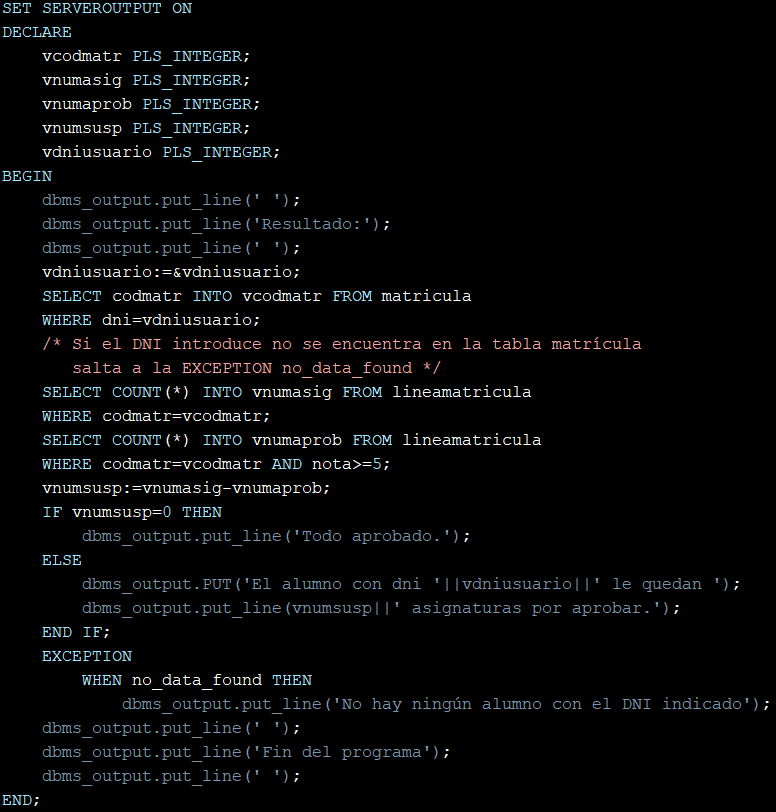
### **9. GESTIÓN DE EXCEPCIONES**

Las acciones sobre la base de datos pueden generar errores SQL o procedimentales en PL/SQL, llamados excepciones. Las excepciones internas de Oracle se activan automáticamente y redirigen la ejecución a la sección **EXCEPTION**.



Se podrán usar tantas cláusulas WHEN como sean necesarias. La opción WHEN OTHERS THEN hace referencia al resto de errores posibles.

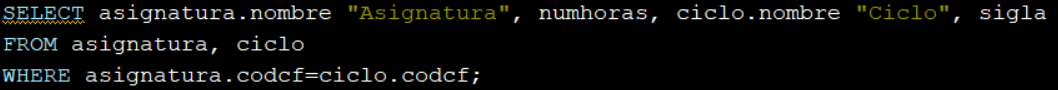
*Vamos a añadir la excepción que avisa si el usuario introduce un DNI que no existe en el ejemplo anterior:*



### **10. COMBINACIÓN DE TABLAS CON JOIN**

El operador **JOIN** permite combinar registros de dos o más tablas de la base de datos.

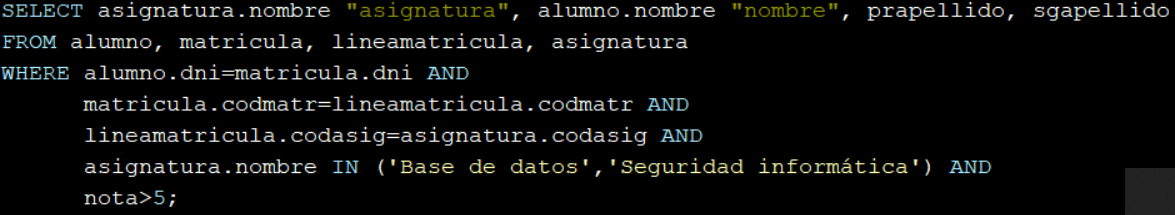
El objetivo es combinar tablas en **FROM** mediante **JOIN**. **USING** cruza columnas con el mismo nombre en varias tablas, evitando ambigüedades si hay nombres repetidos.



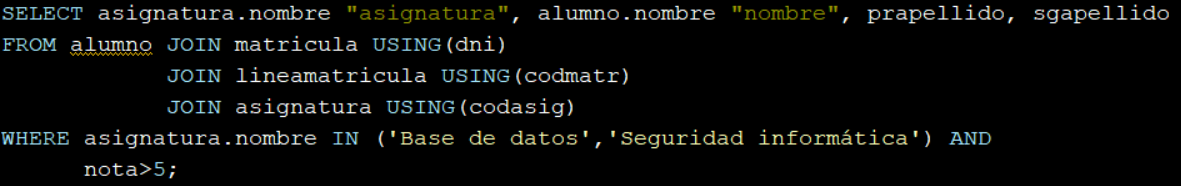
**JOIN USING**:



Si tenemos más de dos tablas:



Usando **JOIN USING** queda:



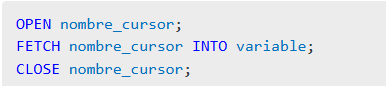
### **11. MANEJO DE CURSORES**

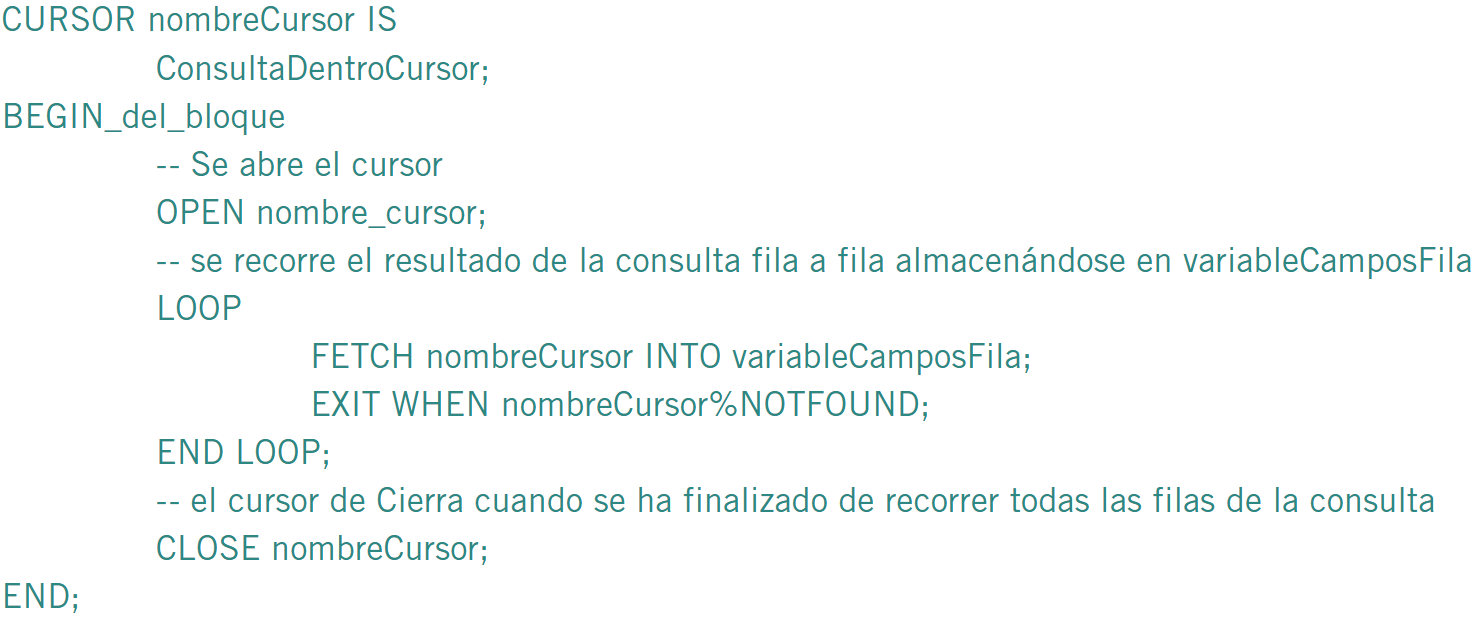
Los **cursores** permiten recorrer las filas de una tabla, o las filas de una consulta, y ejecutar órdenes nuevas para cada una de ellas. Mediante los cursores resolvemos el problema de que la consulta devuelve más de una fila.

#### **11.1. Declaración de un Cursor**

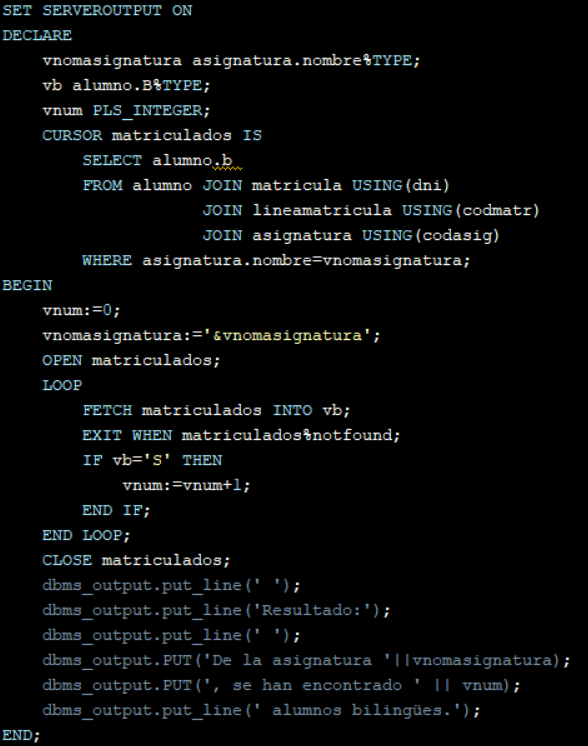
#### 

#### **11.2. Uso de un Cursor**



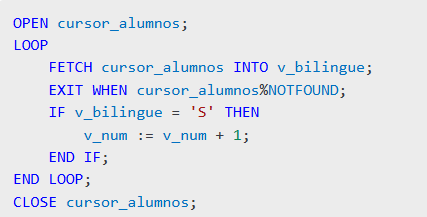


*Por ejemplo, un programa en PL/SQL puede mostrar el número de alumnos bilingües en una asignatura ingresada por teclado*.



**Atributos útiles:**

* **%FOUND**: Devuelve **TRUE** si **FETCH** obtuvo una fila.
* **%NOTFOUND**: Devuelve **TRUE** si **FETCH** no obtuvo una fila.
* **%ROWCOUNT**: Devuelve el número de filas leídas.



### **12. Control mediante Disparadores en PL/SQL**

Los disparadores (triggers) son herramientas que permiten automatizar acciones en una base de datos cuando ocurre un evento sobre una tabla. Su propósito principal es garantizar la integridad de los datos, auditar cambios y comunicar eventos a otros sistemas.

**12.1 Programación de un disparador**

Para programar un disparador, se deben definir dos aspectos clave:

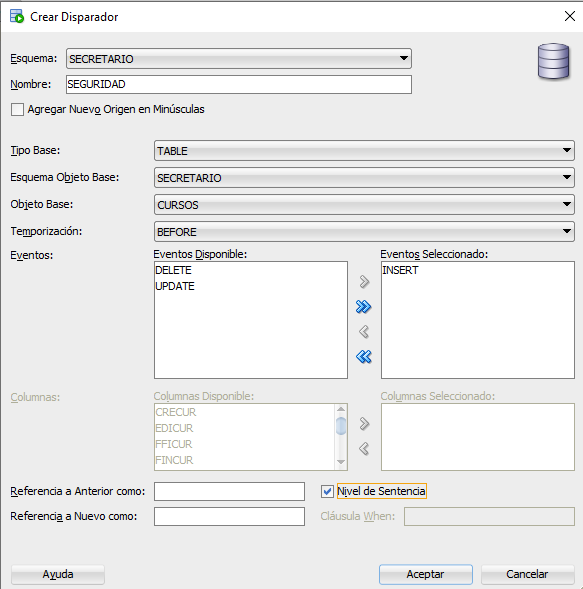
* **Operación a controlar**: **INSERT, DELETE, UPDATE**.
* **Momento de ejecución**: **BEFORE** (antes del evento) o **AFTER** (después del evento).



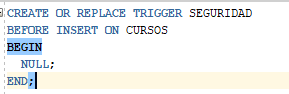
🔹 **Tipos de disparadores:**

* **De comando**: Se ejecutan una única vez por operación.
* **De fila**: Se ejecutan una vez por cada fila afectada.

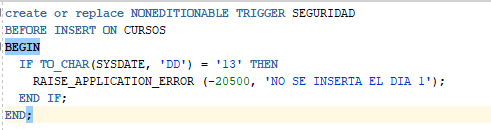
*Ejemplo: diseñar un disparador llamado SEGURIDAD que evite la inserción de filas en la tabla cursos del usuario SECRETARIO si el día del mes es el 1. Disparador a nivel de comando que se ejecutará antes de producirse el evento de inserción.*



Como resultado deberíamos ver:



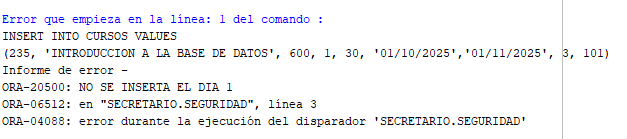
A continuación crearemos el código del disipador



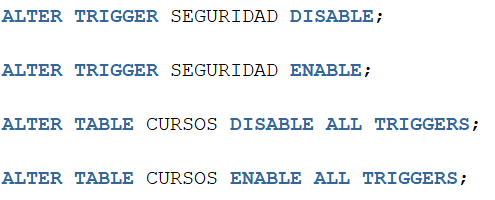
*Ahora vamos a provocar que salte el disparador, como está diseñado para el día 13 del mes, para probarlo cambiamos ‘13’ por el día actual. Ejecutamos en SECRETARIO el siguiente comando para insertar una nueva fila/registro en la tabla CURSOS:*



El disparador salta y nos muestra correctamente el error que hemos indicado:

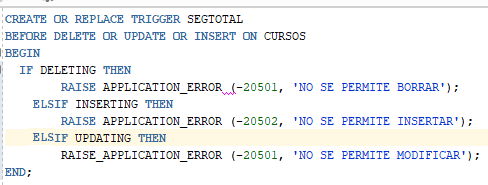


Si deseamos activar/desactivar un disparador mediante comando SQL, utilizaremos los siguientes comandos:

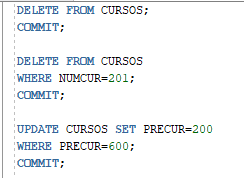


Se pueden combinar varios eventos de un disparador en uno solo, usando los predicados condicionales: **INSERTING**, **DELETING** y **UPDATING**.

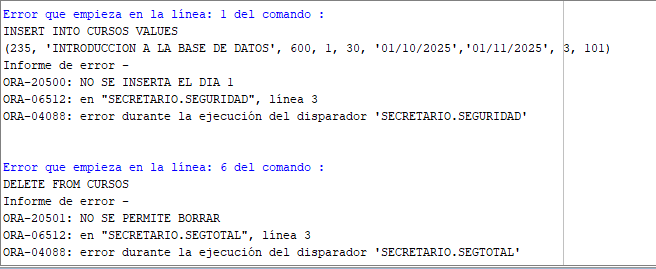
*Ejemplo: diseñar un disparador llamado SEGTOTAL para evitar que se inserte, elimine o modifique una fila/registro de la tabla CURSOS del usuario SECRETARIO:*



Si ejecutamos los comandos:



El disparador se activará en cada caso, mostrándonos el código de error y el mensaje definido en él:



Para consultar información en el diccionario de datos sobre los disparadores se pueden usar las vistas **USER\_TRIGGERS, ALL\_TRIGGERS o DBA\_TRIGGERS**, según corresponda.

*Ejemplo, para conocer toda la información completa sobre los disparadores:*

**SELECT \* FROM DBA\_TRIGGERS;**

*Ejemplo, para conocer la información sobre los disparadores de SECRETARIO:*

**SELECT \* FROM DBA\_TRIGGERS**

**WHERE OWNER='SECRETARIO';**

*Ejemplo, para conocer parte de la información sobre los disparadores de SECRETARIO:*

**SELECT owner, trigger\_name, table\_name,**

**trigger\_type, triggering\_event, status**

**FROM DBA\_TRIGGERS**

**WHERE OWNER='SECRETARIO';**

Para los disparadores de control de filas/registros se usa la cláusula **FOR EACH ROW**. El disparador se ejecuta para cada fila afectada por el evento indicado.

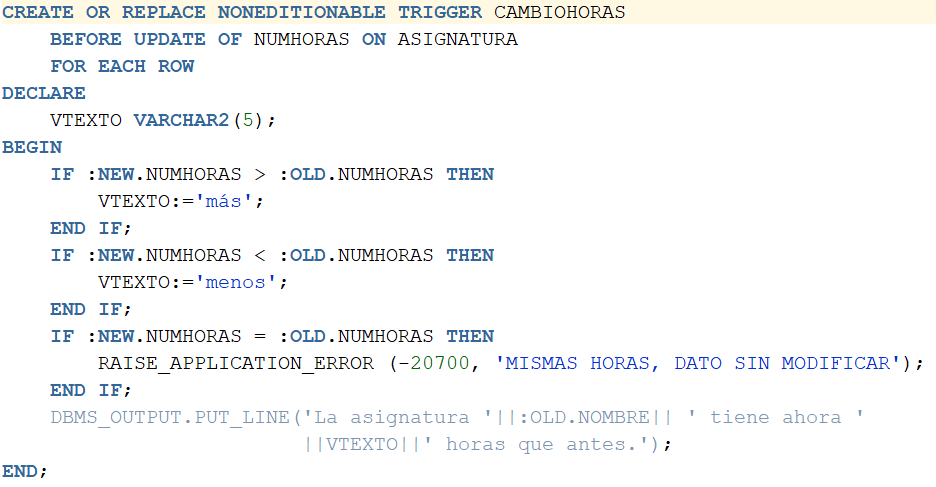
Al programar un disparador para las filas de un comando **INSERT, UPDATE o DELETE** se ejecuta para cada fila.

El primero, **:old** contiene el valor actual sin ejecutar la acción asociada al disparador.

El segundo, **:new** contiene el valor que corresponderá tras la acción del comando.

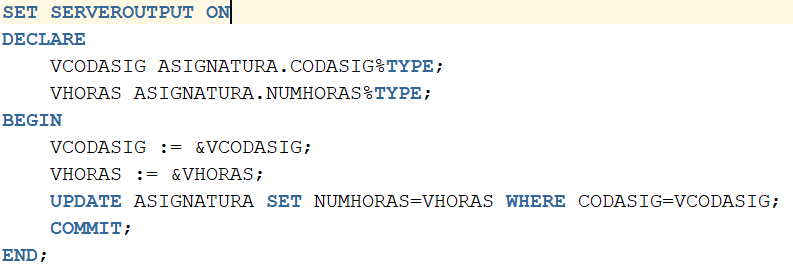
*Ejemplo: diseñar un disparador para la tabla ASIGNATURA del usuario CENTROFOR. El disparador se deberá activar antes que se modifique la columna NUMHORAS de la tabla ASIGNATURA. El disparador deberá mostrar uno de los siguientes mensajes:*

* *La asignatura NOMBREASIGNATURA tiene ahora más horas que antes.*
* *La asignatura NOMBREASIGNATURA tiene ahora menos horas que antes.*



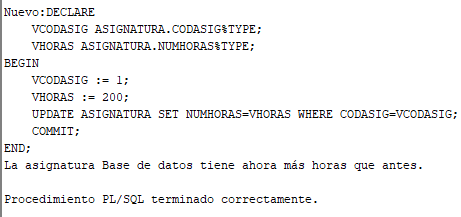
*Para probar el disparador, debemos escribir también un programa PL/SQL que pida el código de la asignatura a modificar, nuevo valor de NUMHORAS y realice la actualización correspondiente.*

**Y el del programa PL/SQL:**



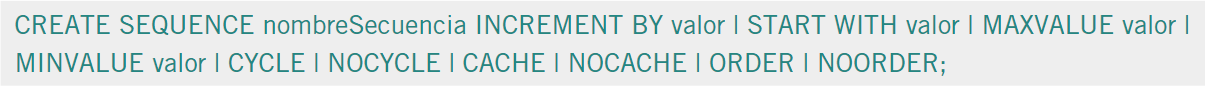
Ejecutamos el código y este nos pedirá dos valores necesarios en los cuales usaremos:

* **VCODASIG**: 1
* **VHORAS**: 200



### **13. Los campos numéricos**

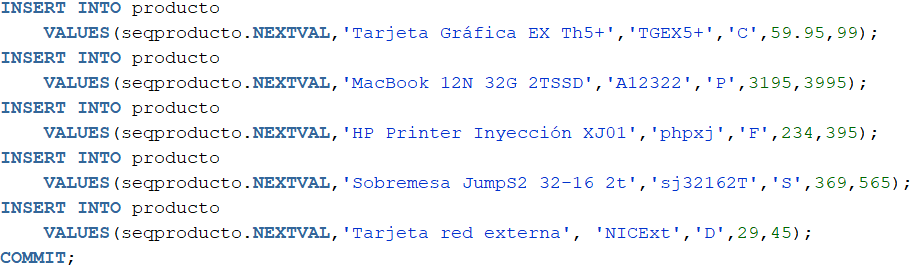
Los SGBD incluyen mecanismos para automatizar este proceso. En Oracle, es necesario crear una secuencia con el comando CREATE SEQUENCE, cuya sintaxis es:



*Por ejemplo, para una supuesta tabla PRODUCTO, el campo primario seqproducto se activaría como autonumérico así:*



Una vez creada la secuencia, se puede usar en los comandos **INSERT**. Para ello, se usará el método **NEXTVAL** que devuelve el valor actual de la secuencia y lo incrementa según el valor especificado en **INCREMENT BY.**



Por último, para comprobar el valor actual de una secuencia, se ejecuta la siguiente consulta:

