**PROCEDIMIENTOS ALMACENADOS**

Son bloques PL-SQL que se almacenan dentro de un objeto persistente en la bbdd.

Los bloques anónimos son códigos que se pierden.

Los procedimientos almacenados permiten guardar la lógica de bloques anónimos dentro de un objeto.

Podemos utilizar cualquier característica vista en bloques anónimos dentro de los procedimientos.

No se pueden utilizar palabras de acción sobre objetos: CREATE, ALTER, DROP.

Los procedimientos pueden tener parámetros. Valores que debemos pasar al procedimiento para poder realizar su ejecución.

Los procedimientos pueden ser dinámicos dependiendo de los parámetros.

Nota: La creación de un procedimiento almacenado, no implica que lo hayamos creado bien.

Sintaxis:

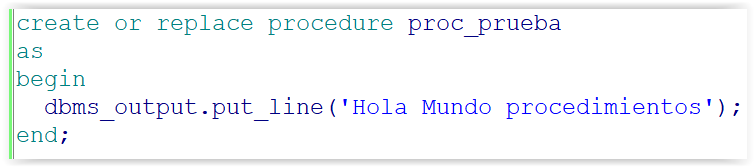
CREATE OR REPLACE PROCEDURE NOMBRE\_PROCEDIMIENTO AS

BEGIN

--INSTRUCCIONES PL-SQL

END;

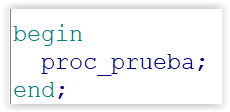
Ejemplo de un procedimiento que muestre un mensaje.



Si estamos en la ventana SQL-PLUS, llamaremos al procedure con **exec**



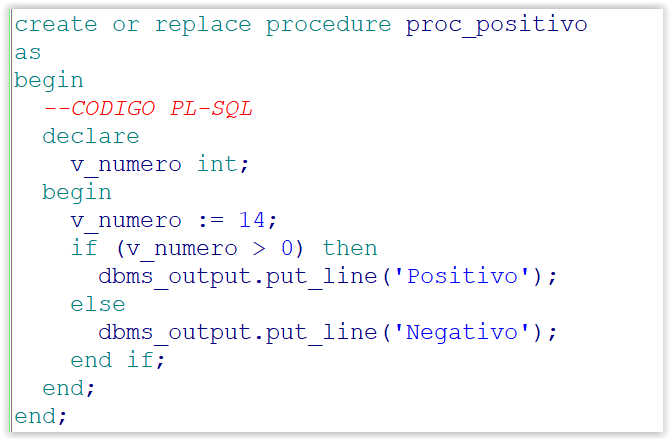
Para llamar a un procedimiento debemos hacerlo desde un bloque PL-SQL con begin y end. También podemos utilizar la palabra exec o execute.



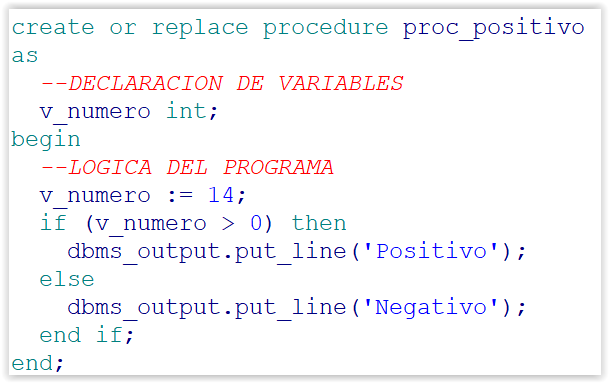
Vamos a crear un procedimiento que contenga código PL-SQL.

Existen dos sintaxis para poder crear un procedimiento con bloques anónimos.

1. Incrustar el código del bloque anónimo dentro del procedimiento, con variables incluidas en el DECLARE



1. Integrar el código PL-SQL dentro del procedimiento. No lleva **declare**



* Detectar errores dentro de un procedimiento almacenado.

Siempre que creemos un nuevo procedimiento, debemos verificar que no contenga errores.

Si el procedimiento tiene errores, debemos solucionarlo antes de poder ejecutarlo, ya que no es válido. El código estará creado en su interior, pero no es ejecutable.

No me avisa de ningún tipo de error a no ser que esté dentro de SQL PLUS.

Si estamos en SQL PLUS, para ver los errores, debemos escribir la instrucción después de crear el procedimiento:

**SQL:> SHOW\_ERRORS;**

**PROCEDIMIENTOS ALMACENADOS CON PARAMETROS**

Un procedimiento con parámetros permite que sus respuestas sean dinámicas.

Los parámetros se denominan con **p\_parametro**.

Los parámetros irán declarados en la cabecera y son obligatorios para poder ejecutar el procedimiento.

Sintaxis:

CREATE OR REPLACE PROCEDURE NOMBRE\_PROCEDIMIENTO

(p\_parametro1 tipodato, p\_parametro2 tipodato)

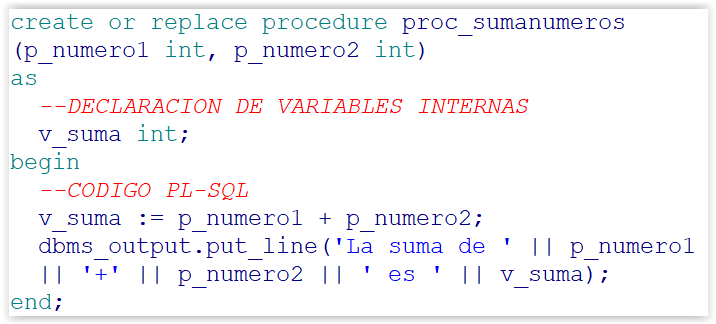
AS

BEGIN

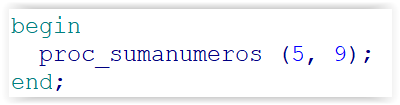
--INSTRUCCIONES PL-SQL

END;

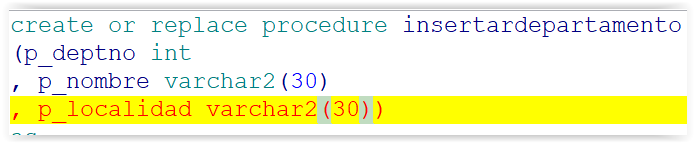
Ejemplo procedimiento sumar dos números enviados como parámetro.



La llamada al procedimiento debemos realizarla dentro de bloques pl-sql y enviando los parámetros dentro de los paréntesis en el orden que hayamos indicado en la cabecera del procedure.



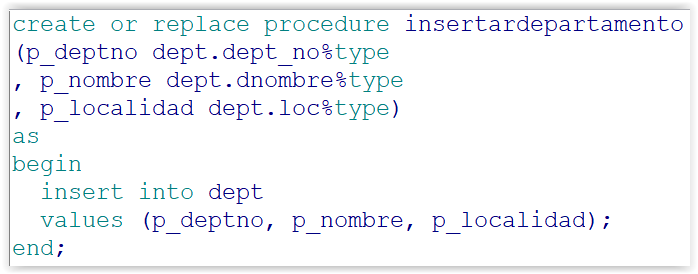
Si declaramos parámetros de tipo STRING (texto) nunca le daremos la longitud dentro de la cabecera de un procedimiento, no nos deja hacerlo.



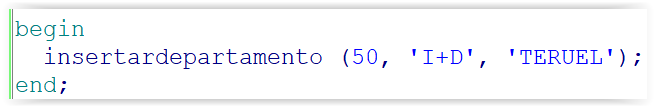
En el tipo de datos, podemos indicar TYPE sin ningún problema, es decir, el tipado de un campo de una tabla para la ejecución.

Ejemplo de procedimiento para insertar un departamento.

Enviaremos todos los datos del departamento.

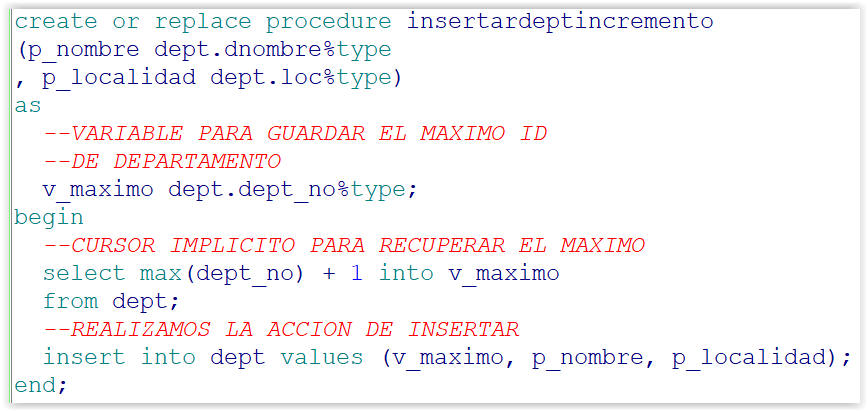


Llamada al procedimiento:



Por supuesto, podemos utilizar tanto cursores explícitos como implícitos en cualquier procedimiento.

Modificar el ejemplo anterior y lo que haremos será recibir solamente dos parámetros (nombre y localidad) y generaremos un autoincremento para el número de departamento mediante un cursor implícito.



Realizar un procedimiento para incrementar el salario de los empleados por un oficio.

Debemos enviar el oficio y el incremento.

**create or replace procedure sp\_incremento\_emp\_oficio**

**(p\_oficio EMP.OFICIO%TYPE, p\_incremento *number*)**

**as**

**begin**

**update EMP set SALARIO = SALARIO + p\_incremento**

**where upper(OFICIO) = upper(p\_oficio);**

**commit;**

**end;**

**begin**

**SP\_INCREMENTO\_EMP\_OFICIO('analista', 1);**

**end;**

Necesito un procedimiento para insertar un doctor.

Enviaremos todos los datos del doctor, excepto el id del doctor

Debemos recuperar el máximo id de doctor dentro del procedimiento

**create or replace procedure sp\_insertar\_doctor**

**(p\_apellido DOCTOR.APELLIDO%TYPE**

**, p\_especialidad DOCTOR.ESPECIALIDAD%TYPE**

**, p\_salario DOCTOR.SALARIO%TYPE**

**, p\_hospital DOCTOR.HOSPITAL\_COD%TYPE)**

**as**

**v\_max\_iddoctor DOCTOR.DOCTOR\_NO%TYPE;**

**begin**

**select max(DOCTOR\_NO) + 1 into v\_max\_iddoctor from DOCTOR;**

**insert into DOCTOR values (p\_hospital, v\_max\_iddoctor**

**, p\_apellido, p\_especialidad, p\_salario);**

**commit;**

**dbms\_output.put\_line('Insertados ' || SQL%ROWCOUNT);**

**end;**

**begin**

**SP\_INSERTAR\_DOCTOR('Willson2', 'Doctor', 280000, 19);**

**end;**

**Versión 2**

Necesito un procedimiento para insertar un doctor.

Enviaremos todos los datos del doctor, excepto el id del doctor

Debemos recuperar el máximo id de doctor dentro del procedimiento

Enviamos **el nombre del hospital** en lugar del id del hospital

Controlar si no existe el hospital enviado

**create or replace procedure sp\_insertar\_doctor**

**(p\_apellido DOCTOR.APELLIDO%TYPE**

**, p\_especialidad DOCTOR.ESPECIALIDAD%TYPE**

**, p\_salario DOCTOR.SALARIO%TYPE**

**, p\_hospital\_nombre HOSPITAL.NOMBRE%TYPE)**

**as**

**v\_max\_iddoctor DOCTOR.DOCTOR\_NO%TYPE;**

**v\_hospitalcod HOSPITAL.HOSPITAL\_COD%TYPE;**

**begin**

**select HOSPITAL\_COD into v\_hospitalcod**

**from HOSPITAL**

**where upper(NOMBRE) = upper(p\_hospital\_nombre);**

**select max(DOCTOR\_NO) + 1 into v\_max\_iddoctor from DOCTOR;**

**insert into DOCTOR values (v\_hospitalcod, v\_max\_iddoctor**

**, p\_apellido, p\_especialidad, p\_salario);**

**dbms\_output.put\_line('Insertados ' || SQL%ROWCOUNT);**

**commit;**

**exception**

**when no\_data\_found then**

**dbms\_output.put\_line('No existe el hospital ' || p\_hospital\_nombre);**

**end;**

**UTILIZAR CURSORES EXPLICITOS DENTRO DE LOS PROCEDIMIENTOS**

Realizar un procedimiento para mostrar los empleados de un determinado número de departamento.

**create or replace procedure sp\_empleados\_dept**

**(p\_deptno EMP.DEPT\_NO%TYPE)**

**as**

**cursor cursor\_emp is**

**select \* from EMP**

**where DEPT\_NO = p\_deptno;**

**begin**

**for v\_reg\_emp in cursor\_emp**

**loop**

**dbms\_output.put\_line('Apellido: ' || v\_reg\_emp.APELLIDO**

**|| ', Oficio: ' || v\_reg\_emp.OFICIO);**

**end loop;**

**end;**

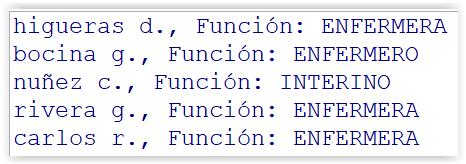
**begin**

**SP\_EMPLEADOS\_DEPT(10);**

**end;**

Ejemplo de procedimiento con parámetros y cursor explícito.

Mostrar el apellido y la función de los empleados de la plantilla de un número de hospital que pasemos como parámetro.



**PROCEDIMIENTOS ALMACENADOS CON PARAMETROS OPCIONALES**

Un procedimiento siempre debe recibir el mismo número de parámetros que hayamos declarado en su cabecera.

Pero los parámetros que creemos pueden ser opcionales si le asignamos un valor por defecto en la declaración.

Si le pasamos valor al parámetro, recuperará el valor enviado. Y no le enviamos valor, tendrá como valor el del defecto.

Sintaxis:

create or replace procedure EjemploOpcionales

(p\_uno int, p\_dos int := 0, p\_tres int := 0)

as

begin

end;

Llamada:

Al ser parámetros opcionales, cualquiera de las opciones es válida.

Begin

EjemploOpcionales (8, 4, 5);

EjemploOpcionales (8, 4);

EjemploOpcionales (8);

End;

Ejemplo cuando los parámetros opcionales son **alternos**

El parámetro p\_tres es obligatorio.

create or replace procedure EjemploOpcionales

(p\_uno int, p\_dos int := 0, p\_tres int)

as

begin

end;

Son incorrectos, porque está esperando el parámetro tres y no se lo estamos indicando.

Begin

EjemploOpcionales (8, 4, 5);

EjemploOpcionales (8, 4);

EjemploOpcionales (8);

End;

Cuando sucede que los parámetros no están al final (recomendable si son opcionales…) debemos indicar el nombre del parámetro junto a su valor utilizando la sintaxis flecha:

**p\_tres => 14**

Begin

EjemploOpcionales (8, 4, 5);

EjemploOpcionales (8, p\_tres => 4);

EjemploOpcionales (8);

End;

Ejemplo:

Pongamos que tenemos un procedimiento que recibe un número y lo multiplica por 2.

El procedimiento siempre multiplicará por dos.

Pero, en ocasiones, queremos que el procedimiento pueda multiplicar por otro valor que le mandemos.

El procedimiento podrá recibir uno o dos parámetros.

Interfaz de usuario gráfica, Aplicación

El contenido generado por IA puede ser incorrecto.

Interfaz de usuario gráfica, Texto

El contenido generado por IA puede ser incorrecto.

**CODIGO SQL**

create or replace procedure multiplicarnumero

(p\_numero int, p\_multiplicador int := 2)

as

v\_resultado int;

begin

v\_resultado := p\_numero \* p\_multiplicador;

dbms\_output.put\_line(p\_numero || '\*' || p\_multiplicador

|| '=' || v\_resultado);

end;

--Llamada

begin

multiplicarnumero (7);

multiplicarnumero (7, 3);

end;

Si los valores son alternos, entonces debemos utilizar flecha.

Vamos a realizar el mismo ejemplo, pero pasaremos tres números.

El segundo número será opcional, que será nuestro multiplicador.

Los otros dos números, mostraremos su suma por el multiplicador.

Interfaz de usuario gráfica, Aplicación

El contenido generado por IA puede ser incorrecto.

Texto

El contenido generado por IA puede ser incorrecto.

**CODIGO SQL**

create or replace procedure multiplicarnumero

(p\_numero1 int, p\_multiplicador int := 2, p\_numero2 int)

as

v\_resultado int;

begin

v\_resultado := (p\_numero1 + p\_numero2) \* p\_multiplicador;

dbms\_output.put\_line('(' || p\_numero1 || '+'

|| p\_numero2 || ') \* '

|| p\_multiplicador

|| ' = ' || v\_resultado);

end;

--Llamada

begin

multiplicarnumero (5, p\_numero2 => 2);

end;

**PROCEDIMIENTOS ALMACENADOS CON PARAMETROS DE SALIDA**

Dentro de un procedimiento almacenado, todos los parámetros son de SOLO LECTURA, por lo que no podremos cambiar su valor si lo deseamos.

En el momento en el que estamos, los procedimientos almacenados de salida no tienen mucho sentido.

Cobran sentido cuando se trabaja en producción, lo que quiere decir con un lenguaje (Java, C#,…) consume un procedimiento de la base de datos.

Los procedimientos, por defecto, reciben parámetros de entrada. Recibimos datos.

Pero los procedimientos también pueden devolver datos en los parámetros.

Un parámetro puede ser entrada o salida, o también puede ser ambos a la vez IN/OUT.

Un **dbms\_output.put\_line()** no puede salir del entorno de Oracle, lo que quiere decir que solamente lo leerá Oracle, nunca ningún otro programa/lenguaje de acceso (Java, C#, …)

Realizar un procedimiento que incremente los empleados de un número de departamento. Enviamos el número y el incremento.

Queremos también tener la suma salarial final.

**JAVA nunca verá la suma salarial**

Texto

El contenido generado por IA puede ser incorrecto.

La solución se encuentra en utilizar parámetros de salida dentro de la cabecera de los procedimientos.

Un parámetro de entrada es un valor (número, texto,…) y un parámetro de salida es una **variable**.

Sintaxis:

create or replace procedure nombre\_procedimiento

(parametro **out** tipodato)

as

begin

end;

Es un valor “extra” que nos puede devolver un procedimiento almacenado.

Ejemplo simple:

Realizar un procedimiento que recibirá un número y nos devolverá su doble. En lugar de utilizar dbms\_output, lo que haremos será devolver el doble en un parámetro.

Texto

El contenido generado por IA puede ser incorrecto.

Llamada:

Texto

El contenido generado por IA puede ser incorrecto.

Además, lo que nos permiten los parámetros de salida es poder enviar cursores entre procedimientos, por ejemplo.

Todo esto, nos permite compartir llamadas y recibir datos entre OBJETOS, por ejemplo, podemos tener un procedimiento que realiza acciones y nos devuelve datos en la cabecera.

Desde otro procedimiento, recuperar dichos datos.

**Ejemplo**

Interfaz de usuario gráfica, Aplicación

El contenido generado por IA puede ser incorrecto.

Ejemplo de un procedimiento que incrementa los

salarios de los empleados de un determinado departamento

y nos devolverá la suma salarial en un parámetro de salida.

**CODIGO SQL**

create or replace procedure incrementarempleados

(p\_incremento int, p\_deptno dept.dept\_no%type

, p\_sumasalarial out int)

as

begin

--MODIFICA LOS DATOS DE LOS EMPLEADOS

--DE UN DEPARTAMENTO

update emp set salario = salario + p\_incremento

where dept\_no = p\_deptno;

select sum(salario) into p\_sumasalarial

from emp

where dept\_no = p\_deptno;

end;

**Llamada**

--LLAMADA JAVA:

declare

v\_suma int;

begin

incrementarempleados (1, 20, v\_suma); --JAVA no ve el dbms\_output

dbms\_output.put\_line('La suma salarial después de update es ' || v\_suma);

end;

**--VAMOS A REALIZAR UN PROCEDIMIENTO PARA ENVIAR EL**

**--NOMBRE DEL DEPARTAMENTO Y DEVOLVER EL NUMERO DE DICHO DEPARTAMENTO**

**create or replace procedure sp\_numerodepartamento**

**(p\_nombre DEPT.DNOMBRE%TYPE, p\_iddept out DEPT.DEPT\_NO%TYPE)**

**as**

**v\_iddept DEPT.DEPT\_NO%TYPE;**

**begin**

**select DEPT\_NO into v\_iddept**

**from DEPT**

**where upper(DNOMBRE) = upper(p\_nombre);**

**p\_iddept := v\_iddept;**

**dbms\_output.put\_line('El número de departamento es ' || v\_iddept);**

**end;**

**begin**

**SP\_NUMERODEPARTAMENTO('ventas');**

**end;**

**--NECESITO UN PROCEDIMIENTO PARA INCREMENTAR EN 1**

**--EL SALARIO DE LOS EMPLEADOS DE UN DEPARTAMENTO.**

**--ENVIAREMOS AL PROCEDIMIENTO EL NOMBRE DEL DEPARTAMENTO**

**create or replace procedure sp\_incrementar\_sal\_dept**

**(p\_nombre DEPT.DNOMBRE%TYPE)**

**as**

**v\_num DEPT.DEPT\_NO%TYPE;**

**begin**

**--recuperamos el id del departamento a partir del nombre**

**--llamamos al procedimiento de numero para recuperar el numero**

**--a partir del nombre**

**--sp\_numerodepartamento**

**--(p\_nombre DEPT.DNOMBRE%TYPE, p\_iddept out DEPT.DEPT\_NO%TYPE)**

**SP\_NUMERODEPARTAMENTO(p\_nombre, v\_num);**

**update EMP set SALARIO = SALARIO + 1**

**where DEPT\_NO=v\_num;**

**dbms\_output.put\_line('Salarios modificados: ' || SQL%ROWCOUNT);**

**end;**

**begin**

**SP\_INCREMENTAR\_SAL\_DEPT('ventas');**

**end;**