

# Administración de Bases de Datos

## Operación y Mantenimiento

### Procedimientos PL/SQL

Los procedimientos en Oracle son objetos que se crean, compilan y ejecutan a petición, y que encapsulan una secuencia de instrucciones para cierta aplicación. Técnicamente se trata de un bloque de código almacenado como un objeto con un nombre dentro de un esquema.

La sintaxis general para crear un procedimiento es:

```
CREATE [OR REPLACE] PROCEDURE nombre_procedimiento
(lista_parámetros)
IS
    declaraciones;
BEGIN
    Sentencias;
EXCEPTION
    Manejo_excepciones;
END [nombre_procedimiento];
```

#### Encabezado del procedimiento

El encabezado del procedimiento especifica su nombre y opcionalmente una lista de parámetros. Cada parámetro puede estar en modo IN, OUT o INOUT, indicando si los parámetros pueden ser de lectura, de escritura o para ambos usos. El modo por omisión es IN, si no hay parámetros no se ponen paréntesis.

Se acostumbra incluir la opción OR REPLACE ya que permite sobre escribir el procedimiento cuando haya que corregir algo y reescribir el código.

#### Cuerpo del procedimiento

El cuerpo del procedimiento tiene tres partes, la parte del código ejecutable es obligatoria y debe tener al menos una sentencia, las otras dos son opcionales.

- Parte de declaraciones. Es donde se declaran las variables, constantes, cursos, etc. No empieza con la palabra DECLARE.
- Parte ejecutable. Contiene una o más sentencias que implementan la aplicación específica, al menos deberá tener la sentencia NULL.
- Parte de manejo de Excepciones. Incluye las condiciones para el manejo de las excepciones.

## Ejemplo

```
SQL> CREATE OR REPLACE PROCEDURE mostrar_contacto(cve NUMBER)
  2  IS
  3      reg_contacto contacto%ROWTYPE;
  4  BEGIN
  5      SELECT * INTO reg_contacto FROM contacto
  6      WHERE clave = cve;
  7      dbms_output.put_line( reg_contacto.nombre || ' ' ||
  8      reg_contacto.apellido || ' <' || reg_contacto.email || '>' );
  9  EXCEPTION
 10      WHEN OTHERS THEN
 11          dbms_output.put_line( SQLERRM );
 12  END;
 13 /
```

Procedimiento creado.

Una vez que se ha compilado con éxito, se puede ejecutar cada vez que sea necesario. La sintaxis para ejecutar un procedimiento es:

`EXECUTE nombre_procedimiento(argumentos)`

O simplemente:

`EXEC nombre_procedimiento(argumentos)`

## Ejemplo

```
SQL> EXEC mostrar_contacto(190);
ORA-01403: No se ha encontrado ningún dato

Procedimiento PL/SQL terminado correctamente.

SQL> EXEC mostrar_contacto(19);
María Contreras  <mary_22@outlook.com>

Procedimiento PL/SQL terminado correctamente.
```

Cuando ya no se requiera conservar un procedimiento almacenado, se emplea el comando `DROP PROCEDURE`, empleando la siguiente sintaxis:

`DROP PROCEDURE nombre_procedimiento;`

## Resultados con sentencias implícitas

La característica de sentencias implícitas permite obtener un conjunto de resultados de un procedimiento almacenado, empleando un cursor del tipo `SYS_REFCURSOR`, el cual es interpretado por la función `dbms_sql.return_result()`.

## Ejemplo

```
SQL> CREATE OR REPLACE PROCEDURE mostrar_contactos(cve NUMBER)
  2  AS
  3      cur_contacto SYS_REFCURSOR;
  4  BEGIN
  5      OPEN cur_contacto FOR
  6          SELECT nombre, apellido, email FROM contacto
  7          WHERE clave >= cve;
  8      dbms_sql.return_result(cur_contacto);
  9  END;
10 /
```

Procedimiento creado.

Este sería el resultado:

```
SQL> EXEC mostrar_contactos(19);

Procedimiento PL/SQL terminado correctamente.
```

Nº de Juego de Resultados 1

NOMBRE	APELLIDO
EMAIL	
Julio j.j66@outlook.com	Jaramillo
Maria mary_22@outlook.com	Contreras

## Funciones PL/SQL

Una función es semejante a un procedimiento, ya que se trata de un programa almacenado que es reutilizable, su sintaxis es la siguiente:

```
CREATE [OR REPLACE] FUNCTION nombre_función (lista_parámetros)
    RETURN tipo_retorno
IS
    Sección_declaraciones
BEGIN
    Sección_ejecutable
[EXCEPTION]
    Sección_de_manejo_de_excepciones
END;
```

En el encabezado de la función se debe indicar el nombre y una cláusula RETURN que especifique el tipo de valor que se retorna, cada parámetro de la lista puede indicarse en los modos IN, OUT o INOUT.

El cuerpo de la función tiene la misma estructura que en los procedimientos. Al menos se debe incluir una sentencia RETURN en la sección ejecutable.

## Ejemplo

```
SQL> CREATE OR REPLACE FUNCTION ventas_totales(eje PLS_INTEGER)
  2  RETURN NUMBER
  3  IS
  4      Total NUMBER := 0;
  5  BEGIN
  6      SELECT SUM(Importe) INTO Total FROM Ventas
  7      WHERE EXTRACT(YEAR FROM Fecha) = eje;
  8      RETURN Total;
  9  END;
10 /
```

Función creada.

Para usar una función almacenada, se debe usar en cualquier expresión que utilice el mismo tipo de dato que el devuelto por la función.

## Ejemplo

```
SQL> DECLARE
  2      ventas_2025 NUMBER := 0;
  3  BEGIN
  4      ventas_2025 := ventas_totales (2025);
  5      DBMS_OUTPUT.PUT_LINE('Ventas totales en 2025: ' || ventas_2025);
  6  END;
  7 /
```

Ventas totales en 2025: 206298

Procedimiento PL/SQL terminado correctamente.

O simplemente como una consulta

```
SQL> SELECT ventas_totales(2025) FROM dual;
```

VENTAS_TOTALES(2025)
-----
206298

Cuando ya no se requiera conservar una función almacenada, se emplea el comando **DROP FUNCTION**, empleando la siguiente sintaxis:

**DROP FUNCTION nombre\_función;**

## Paquetes PL/SQL

Un paquete en PL/SQL, es un esquema que contiene las definiciones de un grupo de elementos tales como: variables, constantes, registros, cursores, excepciones, procedimientos y funciones; y que se compila y almacena en la base de datos Oracle.

Un paquete tiene una especificación y un cuerpo dependiendo de la especificación.

En la especificación se declaran los objetos públicos que se pueden acceder desde fuera del paquete. Para objetos como cursores, el cuerpo del paquete debe contener su definición.

También se pueden declarar variables o cursores privados que solamente utilice el paquete. Incluso puede tener su sección de manejo de excepciones.

Dentro de las ventajas del empleo de paquetes están las siguientes:

- Permite hacer un código modular, encapsulando todos los elementos relacionados lógicamente en módulos, haciendo cada paquete más confiable y reutilizable.
- Oculta los detalles de la implementación, permitiendo la funcionalidad mediante su especificación.
- Mejora el desempeño de las aplicaciones. El paquete se carga en la memoria, por lo que no requiere posteriores accesos al disco.
- Minimiza recompilación innecesaria de bloques de código. Si se modifica una función dentro del paquete, Oracle no tiene que recompilar ningún otro programa que utilice dicha función.
- Facilita el control de autorizaciones. Al encapsular los objetos del paquete, se otorgan permisos sobre el paquete, en vez de otorgarlos sobre los objetos contenidos en el paquete.

### Especificación de los Paquetes PL/SQL

Un paquete tiene dos partes, la especificación y el cuerpo, en la especificación se declaran los objetos públicos y por omisión, su ámbito es el esquema que contiene al paquete.

La especificación no contiene la implementación de sus elementos, es decir, solamente contiene las cabeceras de procedimientos y funciones, no sus cuerpos.

Para crear un paquete se emplea la siguiente sintaxis:

```
CREATE [OR REPLACE] PACKAGE [esquema.]nombre_paquete IS | AS
    declaraciones;
END [nombre_paquete];
```

## Ejemplo

```
SQL> CREATE OR REPLACE PACKAGE control_pedidos
  2 AS
  3   status_enviado CONSTANT VARCHAR(10) := 'Enviado';
  4   status_pendiente CONSTANT VARCHAR(10) := 'Pendiente';
  5   status_cancelado CONSTANT VARCHAR(10) := 'Cancelado';
  6   total NUMBER;
  7
  8   CURSOR cur_ordenes(eje NUMBER) IS
  9     SELECT * FROM Ventas WHERE EXTRACT(YEAR FROM Fecha) = eje;
 10
 11  FUNCTION importe_pedido(num_pedido NUMBER)
 12    RETURN NUMBER;
 13  FUNCTION importe_cliente(cve_cliente NUMBER)
 14    RETURN NUMBER;
 15 END control_pedidos;
 16 /
```

Paquete creado.

Para referirse a algún elemento dentro del paquete se emplea la siguiente sintaxis:

`nombre_paquete.nombre_elemento`

## Ejemplo

```
SQL> BEGIN
  2   DBMS_OUTPUT.PUT_LINE(control_pedidos.status_pendiente);
  3 END;
 4 /
Pendiente

Procedimiento PL/SQL terminado correctamente.
```

## Cuerpo del Paquete

Cuando el paquete incluye cursos o subprogramas, es necesario incluir el cuerpo del paquete. Cada cursor o subprograma que se declare en el paquete debe tener su definición correspondiente en el cuerpo del programa. Adicionalmente podrá contener elementos privados adicionales.

Para crear el cuerpo del paquete se emplea el comando CREATE PACKAGE BODY como se muestra:

```
CREATE [OR REPLACE] PACKAGE BODY esquema.]nombre_paquete IS | AS
  declaraciones
  implementaciones;
[BEGIN/EXCEPTION]
END [nombre_paquete];
```

## Ejemplo

```
SQL> CREATE OR REPLACE PACKAGE BODY control_pedidos
  2  AS
  3      FUNCTION importe_pedido(num_pedido NUMBER)
  4          RETURN NUMBER
  5      IS
  6          imp NUMBER := 0;
  7      BEGIN
  8          SELECT Importe INTO imp FROM Ventas
  9          WHERE Folio = num_pedido;
 10         RETURN imp;
 11     EXCEPTION
 12         WHEN no_data_found THEN
 13             DBMS_OUTPUT.PUT_LINE( SQLERRM );
 14     END importe_pedido;
 15
 16     FUNCTION importe_cliente(cve_cliente NUMBER)
 17         RETURN NUMBER
 18     IS
 19         Total NUMBER := 0 ;
 20     BEGIN
 21         SELECT SUM(Importe) INTO Total FROM Ventas
 22         WHERE Cliente = cve_cliente;
 23         RETURN Total;
 24     EXCEPTION
 25         WHEN no_data_found THEN
 26             DBMS_OUTPUT.PUT_LINE( SQLERRM );
 27     END importe_cliente;
 28 END control_pedidos;
 29 /
```

Cuerpo del paquete creado.

Compilar el cuerpo del paquete es similar a compilar la especificación del paquete, a continuación, ya se pueden ejecutar las funciones definidas en el paquete.

## Ejemplo

```
SQL> SELECT Control_pedidos.importe_cliente(5) "Importe Total" FROM dual;
Importe Total
_____
138408
```

Para eliminar un paquete se emplea la sentencia DROP PACKAGE, con la siguiente sintaxis:

**DROP PACKAGE [BODY] [nombre\_esquema].nombre\_paquete;**

Se incluye la declaración BODY, si solamente se desea eliminar el cuerpo del paquete, en caso contrario se eliminan ambas partes del paquete.

## Triggers en Oracle

Un disparador o trigger es un bloque PL/SQL almacenado en la base de datos Oracle y ejecutado automáticamente cuando un evento disparador ocurre.

Los disparadores son útiles en casos como los siguientes:

- Prevención de transacciones inválidas.

- Reforzamiento de reglas de negocio complejas que no se pueden limitar al establecer restricciones de integridad simples como UNIQUE, NOT NULL o CHECK.
- Obtención de información estadística sobre el acceso a tablas.
- Calcular valores de columnas derivadas en forma automática.
- Auditarse los datos más críticos.

El evento detonante puede ser alguno de éstos:

- Cualquier sentencia DML ejecutada sobre una tabla, como son INSERT, UPDATE o DELETE.
- Cualquier sentencia DDL como: CREATE or ALTER. Este tipo de disparadores se utilizan generalmente con propósitos de monitoreo y auditoría de los cambios que se realizan en los esquemas.
- Eventos del sistema, tales como STARTUP o SHUTDOWN de la base de datos.
- Eventos de usuarios, tales como LOGIN o LOGOUT.

## Creación de disparadores

Para crear un nuevo disparador se emplea la sentencia CREATE TRIGGER, con la siguiente sintaxis:

```
CREATE [OR REPLACE] TRIGGER nombre_disparador
{BEFORE | AFTER} evento_detonante ON nombre_tabla
[FOR EACH ROW]
[FOLLOWS | PRECEDES otro_disparador]
[ENABLE / DISABLE]
[WHEN condición]
DECLARE
    declaraciones
BEGIN
    Sentencias_ejecutables
EXCEPTION
    manejo_excepciones
END;
```

Los elementos del encabezado del disparador incluyen diversas especificaciones importantes, mientras que el cuerpo del disparador es como un bloque de código PL/SQL normal.

Utilizar CREATE OR REPLACE permite reemplazar un disparador existente o crear uno nuevo en caso contrario, sin provocar errores.

Las opciones BEFORE o AFTER especifican si el disparador se detona antes o después del evento detonante ocurrido en la tabla asociada.

La sentencia FOR EACH ROW especifica que se trata de un disparador a nivel de registros, es decir, que se acciona para cada registro modificado.

En caso de faltar esta sentencia, se considera que se trata de un disparador de nivel sentencia, y solamente se acciona una sola vez sin importar el número de registros afectados por el evento detonante.

Para un determinado evento se pueden definir varios disparadores, entonces se debe determinar una secuencia usando las indicaciones **FOLLOW**s o **PRECEDES** con otro disparador.

Desde el momento que se crea un disparador queda habilitado (**ENABLE**), pero si se declara **DISABLE**, no actuará cuando ocurra un evento detonante.

La cláusula **WHEN** se emplea para mejorar la eficiencia del disparador declarando las condiciones específicas en que debe actuar.

### Ejemplo

Suponga que se tiene una tabla donde se registran los cambios a la tabla de clientes, mediante el siguiente disparador sobre los eventos **UPDATE** y **DELETE**.

```
SQL> CREATE OR REPLACE TRIGGER cambios_clientes
  2  BEFORE DELETE OR UPDATE ON "Cliente"
  3  FOR EACH ROW
  4  DECLARE
  5    ope VARCHAR2(10);
  6  BEGIN
  7    ope := CASE
  8      WHEN UPDATING THEN 'UPDATE'
  9      WHEN DELETING THEN 'DELETE'
 10  END;
 11  INSERT INTO cambios (TABLA,OPERACION,USUARIO,FECHA)
 12  VALUES('Cliente', ope, USER, SYSDATE);
 13 END;
 14 /
```

Disparador creado.

Entonces, si se ejecuta la siguiente sentencia de actualización y luego se checa el contenido de la tabla Cambios, se observa el registro generado automáticamente:

```
SQL> UPDATE "Cliente" SET Credito =20000 WHERE "Clave"=15;
```

1 fila actualizada.

```
SQL> SELECT * FROM Cambios;
```

TABLA	OPERACION	USUARIO	FECHA
Cliente	UPDATE	C##VENTAS	10/04/25

Algo semejante ocurre cuando se elimina algún registro:

```
SQL> DELETE FROM "Cliente" WHERE "Clave"=20;
```

1 fila suprimida.

```
SQL> SELECT * FROM Cambios;
```

TABLA	OPERACION	USUARIO	FECHA
Cliente	UPDATE	C##VENTAS	10/04/25
Cliente	DELETE	C##VENTAS	10/04/25

## Disparadores a nivel de sentencia

Los disparadores a nivel de sentencia tienen una finalidad de permitir o impedir la ejecución de alguna sentencia dependiendo de las circunstancias en las que se realice. Por ejemplo, suponga que por razones contables esté prohibido modificar el crédito de un cliente después del día 25 de cada mes. Esta regla se puede implementar mediante un disparador de nivel sentencia.

### Ejemplo

```
SQL> CREATE OR REPLACE TRIGGER cambio_credito
  2    BEFORE UPDATE OF Credito ON "Cliente"
  3    DECLARE
  4      dia NUMBER;
  5    BEGIN
  6      dia := EXTRACT(DAY FROM sysdate);
  7      IF dia > 25 THEN
  8        raise_application_error(-20100,'No puede cambiar el crédito de un cliente después del día 25');
  9      END IF;
10    END;
11  /
```

Disparador creado.

Si se ejecuta una sentencia de actualización de crédito después del día 25, el disparador genera una excepción que impide los cambios.

```
SQL> UPDATE "Cliente" SET Credito =200000 WHERE "Clave">>=25;
UPDATE "Cliente" SET Credito =200000 WHERE "Clave">>=25
*
ERROR en línea 1:
ORA-20100: No puede cambiar el crédito de un cliente después del día 25
ORA-06512: en "C##VENTAS.CAMBIO_CREDITO", línea 6
ORA-04088: error durante la ejecución del disparador 'C##VENTAS.CAMBIO_CREDITO'
```

Oracle efectúa un ROLLBACK automáticamente como consecuencia de la llamada a RAISE\_APPLICATION\_ERROR en el trigger.

## Disparadores a nivel de registro

Los disparadores a nivel registro se disiran para cada uno de los registros afectados y suelen ser muy útiles para validación de datos y auditoría. En estos disparadores se debe declarar la cláusula FOR EACH ROW.

Una característica importante de estos disparadores es que permiten el acceso a los valores anteriores y nuevos de cada columna de los registros afectados. Para ello se utilizan los prefijos :OLD y :NEW antes del nombre de la columna. Estos valores están disponibles dependiendo del evento ocurrido, en una inserción no existe el valor anterior (:OLD), mientras que en una eliminación no existe un valor nuevo (:NEW). Solamente en las actualizaciones existen ambos valores. Dentro de una cláusula WHEN no se empean los dos puntos antes de OLD o NEW.

Además, si el disparador es anterior al evento (BEFORE) puede modificar los valores nuevos antes de registrarlos.

## Ejemplo

```
SQL> CREATE OR REPLACE TRIGGER cambio_credito2
  2      BEFORE UPDATE OF Credito ON "Cliente"
  3      FOR EACH ROW
  4      WHEN (NEW.Credito > 0)
  5  BEGIN
  6      IF :NEW.Credito >= 2 * :OLD.Credito THEN
  7          raise_application_error(-20101,'El nuevo crédito ' || :NEW.Credito || ' no puede
incrementar más del doble, el crédito actual es: ' || :OLD.Credito);
  8      END IF;
  9  END;
10 /
```

Disparador creado.

Como se pretende un incremento mayor al doble, el disparador lo evita:

```
SQL> UPDATE "Cliente" SET Credito=50000 WHERE "Clave"= 41;
UPDATE "Cliente" SET Credito=50000 WHERE "Clave"= 41
*
ERROR en línea 1:
ORA-20101: El nuevo crédito 50000 no puede incrementar más del doble, el
crédito actual es: 20000
ORA-06512: en "C##VENTAS.CAMBIO_CREDITO2", linea 3
ORA-04088: error durante la ejecución del disparador
'C##VENTAS.CAMBIO_CREDITO2'
```

## Disparadores INSTEAD OF

Un disparador INSTEAD OF es aquel que permite actualizar datos en tablas a través de una vista que no puede ser modificada directamente con comandos DML.

La sintaxis de este tipo de disparadores es:

```
CREATE [OR REPLACE] TRIGGER nombre_disparador
INSTEAD OF {INSERT | UPDATE | DELETE}
ON nombre_vista
FOR EACH ROW
BEGIN
    EXCEPTION
    ...
END;
```

## Ejemplo

Se tiene una vista llamada Vista1, con datos de ventas por cliente, es imposible insertar un nuevo cliente directamente a través de esta vista:

```
SQL> INSERT INTO Vista1 VALUES (4,'11/04/25',3000,'Laura Rosas',2000);
INSERT INTO Vista1 VALUES (4,'11/04/25',3000,'Laura Rosas',2000)
*
ERROR en línea 1:
ORA-01776: no se puede modificar más de una tabla base a través de una vista de
unión
```

Pero es posible insertar los registros necesarios de cliente y venta, si se crea un disparador INSTEAD OF adecuado:

```
SQL> CREATE OR REPLACE TRIGGER nuevo_cliente
  2      INSTEAD OF INSERT ON Vista1
  3          FOR EACH ROW
  4  DECLARE
  5      clave_cte NUMBER;
  6  BEGIN
  7      INSERT INTO "Cliente" ("Nombre",credito)
  8          VALUES(:NEW.Cliente, :NEW.Credito)
  9      RETURNING "Clave" INTO clave_cte;
10      INSERT INTO Ventas (Folio, Fecha, Importe, Cliente)
11          VALUES(:NEW.Folio, :NEW.Fecha, :NEW.Importe, clave_cte);
12  END;
13 /
```

Disparador creado.

Mediante el disparador se pudo insertar al nuevo cliente, obtener su clave e insertar el folio de la venta.

```
SQL> INSERT INTO Vista1 VALUES (4,'11/04/25', 3000,'Laura Rosas',2000);
1 fila creada.
```

### Deshabilitar y habilitar disparadores

A veces se desea deshabilitar un disparador para realizar pruebas o resolver alguna falla, para ello se emplea la siguiente sentencia:

```
ALTER TRIGGER nombre_disparador DISABLE;
```

Para habilitar un disparador se ejecuta una sentencia similar:

```
ALTER TRIGGER nombre_disparador ENABLE;
```

Otra alternativa útil es la de deshabilitar todos los disparadores ligados a alguna tabla, mediante la siguiente sentencia:

```
ALTER TABLE nombre_tabla DISABLE ALL TRIGGERS;
```

De forma semejante se pueden habilitar todos los disparadores asociados con una tabla:

```
ALTER TABLE nombre_tabla ENABLE ALL TRIGGERS;
```

Para eliminar un disparador se emplea la sentencia:

```
DROP TRIGGER [nombre_esquema.]nombre_disparador;
```