

**MP0484. Bases de datos
UF7. Programación avanzada
de acceso a datos**

7.3. Disparadores

Índice

☰	Introducción y objetivos	3
☰	Definición de disparador	4
☰	Disparadores de tablas	5
☰	Disparadores de sustitución	13
☰	Disparadores del sistema	15
☰	Consideraciones sobre disparadores	19
☰	Excepciones	21
☰	Ejercicio resuelto	24

Introducción y objetivos

Los disparadores en base de datos permiten poder realizar acciones programáticas en el caso de que se produzcan determinados eventos en una base de datos.

Es una herramienta que permite de alguna forma controlar todo el funcionamiento de la base de datos para que sea estable, bien construida y mantenida. Hay que tener en cuenta que muchos usuarios y programas accederán a los datos inmersos en la base de datos y los TRIGGER nos permiten controlarla de alguna forma.

Objetivos:

- Conocer cómo se crean los disparadores.
- Conocer los usos de los disparadores en bases de datos.
- Conocer y manejar los Disparadores DML.
- Usar los nombres de correlación y pseudorecords.
- Usar los Disparadores de Sistema.
- Manejar los errores y las excepciones en los disparadores.
- Conocer las buenas prácticas para usar disparadores.
- Manejar el orden de ejecución de disparadores.
- Conocer la habilitación y deshabilitación de disparadores.

Definición de disparador

Por lo tanto un disparador o TRIGGER es un código específico que puede invocarse o lanzarse más de una vez. Un disparador se puede activar y desactivar de forma que pueda ser llamado o no por un elemento programático.

La peculiaridad de un disparador en relación a los procedimientos almacenados es que el disparador es invocado cuando un evento de la base de datos ocurre, siempre que esté activado.

Hay tres tipos de disparadores de bases de datos:

- **Disparadores de tablas.** Asociados a una tabla. Se disparan cuando se produce un determinado suceso o evento de manipulación que afecta a la tabla (inserción, borrado o modificación de filas).
- **Disparadores de sustitución.** Asociados a vistas. Se disparan cuando se intenta ejecutar un comando de manipulación que afecta a la vista (inserción, borrado o modificación de filas).
- **Disparadores del sistema.** Se disparan cuando ocurre un evento del sistema (arranque o parada de la base de datos, entrada o salida de un usuario, etcétera) o una instrucción de definición de datos (creación, modificación o eliminación de una tabla u otro objeto).

Usos de los disparadores

Se pueden utilizar para:

- Implementar restricciones complejas de seguridad o integridad.
- Posibilitar la realización de operaciones de manipulación sobre vistas.
- Prevenir transacciones erróneas.
- Implementar reglas administrativas complejas.
- Generar automáticamente valores derivados.
- Auditlar las actualizaciones e, incluso, enviar alertas.
- Gestionar réplicas remotas de la tabla.
- Etc.

Disparadores de tablas

Los disparadores de tablas son disparadores asociados a una determinada tabla de la base de datos.

Se disparan cuando se produce un determinado suceso o evento de manipulación que afecta a la tabla (inserción, borrado o modificación de filas).

El siguiente ejemplo crea el trigger audit_subida_salario, que se disparará después de cada modificación de la columna salario de la tabla employees de la base de datos HR de ejemplo de Oracle. En la tabla auditareemple tendremos auditadas todas las subidas de salarios realizadas:

```
CREATE OR REPLACE TRIGGER audit_subida_salario
  AFTER UPDATE OF salary ON employees
  FOR EACH ROW
  BEGIN
    INSERT INTO auditareemple
      VALUES ('SUBIDA SALARIO EMPLEADO ' || :old.employee_id );
  END;
  /
```

Este disparador requiere de la tabla auditareemple, que habrá sido creada:

```
CREATE TABLE auditareemple (log VARCHAR2(200));
```

De esta manera si ejecutamos la sentencia:

```
UPDATE employees SET salary=18000 WHERE employee_id=100;
```

Crearemos registros del siguiente tipo en la tabla auditareemple:

```
SUBIDA SALARIO EMPLEADO 100
```

Formato para la creación de disparadores de tablas

El formato es el siguiente:

```
CREATE [OR REPLACE] TRIGGER nombretrigger
    {BEFORE | AFTER}
    {DELETE | INSERT | UPDATE [OF <lista_columnas>]}
    ON nombretabla
        [FOR EACH {STATEMENT | ROW      [WHEN (condicion)]}]
        < CUERPO DEL TRIGGER (BLOQUE PL/SQL)>
```

Hay que destacar las siguientes cuestiones:

- El evento de disparo será una orden de manipulación: INSERT, DELETE o UPDATE. En el caso de esta última, se podrán especificar opcionalmente las columnas cuya modificación producirá el disparo.
- El momento en que se ejecuta el trigger puede ser antes (BEFORE) o después (AFTER) de que se ejecute la orden de manipulación.
- El nivel de disparo del trigger puede ser a nivel de orden o a nivel de fila.

- A nivel de orden (STATEMENT). El trigger se activará una sola vez para cada orden, independientemente del número de filas afectadas por ella. Se puede incluir la cláusula FOR EACH STATEMENT, aunque no es necesario, ya que se asume por omisión.
- A nivel de fila (ROW): el disparador se activará una vez para cada fila afectada por la orden.
- La restricción del trigger. La cláusula WHEN seguida de una condición restringe la ejecución del trigger al cumplimiento de la condición especificada. Esta condición tiene algunas limitaciones:
 - Solamente se puede utilizar con triggers a nivel de fila (FOR EACH ROW).
 - Se trata de una condición SQL, no PL/SQL.
 - No puede incluir una consulta a la misma o a otras tablas o vistas.

Ejemplo:

El siguiente ejemplo crea un trigger que se disparará cada vez que se borre un empleado, guardando su número de empleado, apellido y departamento en una fila de la tabla auditaremple:

```
CREATE OR REPLACE TRIGGER audit_borrado_emple
BEFORE DELETE ON employees
FOR EACH ROW
BEGIN
    INSERT INTO auditaremple
    VALUES ('BORRADO EMPLEADO' || '*' || :old.employee_id || '*' || :old.first_
    || '*Dpto.' || :old.department_id);
END;
/
```

Si quisiéramos incluir una restricción para que sólo se ejecute el disparador cuando el empleado borrado sea el PRESIDENTE lo indicaremos insertando una cláusula WHEN antes del cuerpo del trigger:

```
WHEN (old.job_id = 'AD_PRES')
```

Valores NEW y OLD

Se puede hacer referencia a los valores anterior y posterior a una actualización a nivel de fila.

Lo haremos como :old.nombrecolumna y :new.nombrecolumna respectivamente.

Por ejemplo:

```
... IF :new.salary < :old.salary ...
```

Al utilizar los valores old y new deberemos tener en cuenta el evento de disparo:

- Cuando el evento que dispara el trigger es DELETE, deberemos hacer referencia a :old.nombrecolumna, ya que el valor de new es NULL.
- Paralelamente, cuando el evento de disparo es INSERT, deberemos referirnos siempre a :new.nombrecolumna, puesto que el valor de old no existe (es NULL).
- Para los triggers cuyo evento de disparo es UPDATE, tienen sentido los dos valores.

En el caso de que queramos hacer referencia a los valores new y old, al indicar la restricción del trigger (en la cláusula WHEN), lo haremos sin poner los dos puntos.

Por ejemplo:

```
WHEN new.salary < old.salary
```

Orden de ejecución de los disparadores

Una misma tabla puede tener varios disparadores. El orden de disparo será el siguiente:

- Antes de comenzar a ejecutar la orden que produce el disparo, se ejecutarán los disparadores BEFORE ... FOR EACH STATEMENT.
- Para cada fila afectada por la orden:
 1. Se ejecutan los disparadores BEFORE ... FOR EACH ROW.
 2. Se ejecuta la actualización de la fila (INSERT UPDATE o DELETE). En este momento se bloquea la fila hasta que la transacción se confirme.
 3. Se ejecutan los disparadores AFTER ... FOR EACH ROW.
- Una vez realizada la actualización se ejecutarán los disparadores AFTER ... FOR EACH STATEMENT.

En el caso de que existan varios del mismo tipo y queramos controlar su ejecución se pueden usar las cláusulas **FOLLOW**s y **PRECEDE**s. Estas cláusulas especifican la relación en el lanzamiento de TRIGGER que tienen el mismo punto de temporización. Es especialmente útil cuando se crean TRIGGER, que se deben disparar en un orden específico para lograr su propósito. Se debe usar **FOLLOW**s para indicar que el TRIGGER que se está definiendo y creando debe lanzarse después de otro. Se debe usar **PRECEDE**s para indicar que el TRIGGER que se está creando se debe lanzar antes de otro.



OBSERVACIONES

- Cuando se dispara un trigger, éste forma parte de la operación de actualización que lo disparó, de manera que si el trigger falla, Oracle dará por fallida la actualización completa. Aunque el fallo se produzca a nivel de una sola fila, Oracle hará ROLLBACK de toda la actualización.
- En ocasiones, en lugar de asociar varios triggers a una tabla, podremos optar por la utilización de un solo trigger con múltiples eventos de disparo, tal como se explica en el apartado siguiente.

Disparadores compuestos

Un disparador o TRIGGER DML puede ser simple o compuesto.

- Los simples sólo trabajan con un evento de activación.
- Los disparadores compuestos o Compound Triggers se crean para poder combinar los cuatro eventos de activación en una sola pieza de código.

La sintaxis sería:

```
CREATE OR REPLACE TRIGGER nombreTriggerCompuesto
FOR INSERT | DELETE UPDATE OF column ON table COMPOUND TRIGGER
```

A continuación se muestra un ejemplo:

```

CREATE OR REPLACE TRIGGER nombreDisparadorCompuesto FOR UPDATE
OF salario ON empleados
COMPOUND TRIGGER

    -- Parte declarativa
    -- Variables declaradas para la ejecución.

    -- Ejecución antes de una consulta DML
BEFORE STATEMENT IS
BEGIN
    NULL;
END BEFORE STATEMENT;

    -- Ejecución antes de cada fila, variables :NEW, :OLD son permitidas
BEFORE EACH ROW IS
BEGIN
    NULL;
END BEFORE EACH ROW;

    -- Ejecución después de cada fila, variables :NEW, :OLD son permitidas
AFTER EACH ROW IS
BEGIN
    NULL;
END AFTER EACH ROW;

    -- Ejecución después de una consulta DML
AFTER STATEMENT IS
BEGIN
    NULL;
END AFTER STATEMENT;
END nombreDisparadorCompuesto ;

```

Múltiples eventos de disparo y predicados condicionales

Un mismo trigger puede ser disparado por distintas operaciones o eventos de disparo. Para indicarlo, se utilizará el operador OR.

Por ejemplo:

```
... BEFORE DELETE OR UPDATE ON employees ...
```

En estos casos, es probable que una parte de las acciones del bloque PL/SQL dependa del tipo de evento que disparó el trigger. Para facilitar este control Oracle permite la utilización de predicados condicionales que devolverán un valor verdadero o falso para cada una de las posibles operaciones:

INSERTING

Devuelve TRUE si el evento que disparó el trigger fue un comando INSERT.

DELETING

Devuelve TRUE si el evento que disparó el trigger fue un comando DELETE.

UPDATING

Devuelve TRUE si el evento que disparó el trigger fue una instrucción UPDATE.

UPDATING ('nombrecolumna')

Devuelve TRUE si el evento que disparó el trigger fue una instrucción UPDATE y la columna especificada ha sido actualizada.

Ejemplo:

```
CREATE TRIGGER ...
BEFORE INSERT OR UPDATE OR DELETE ON empleados ...
BEGIN
  IF INSERTING THEN
    ...
  ELSIF DELETING THEN
    ...
  ELSIF UPDATING('salary') THEN
    ...
  END IF
  ...
END;
```

Restricciones

El código PL/SQL del cuerpo del trigger puede contener instrucciones de consulta y de manipulación de datos, así como llamadas a otros subprogramas. No obstante, existen restricciones que deben ser contempladas:

- El bloque PL/SQL no puede contener sentencias de control de transacciones como COMMIT, ROLLBACK o SAVEPOINT.
- Tampoco se pueden hacer llamadas a procedimientos que transgredan la restricción anterior.
- No se pueden utilizar comandos DDL.
- Un trigger no puede contener instrucciones que consulten o modifiquen tablas mutantes. Una tabla mutante es aquella que está siendo modificada por una sentencia UPDATE, DELETE o INSERT en la misma sesión.
- No se pueden cambiar valores de las columnas que sean claves primaria, única o ajena de tablas de restricción. Una tabla de restricción es una tabla que debe ser consultada o actualizada directa o indirectamente por el comando que disparó el trigger (normalmente, debido a una restricción de integridad referencial) en la misma sesión.

Los triggers a nivel de comando (FOR EACH STATEMENT) no se verán afectados por las restricciones que acabamos de enunciar para las tablas mutantes y tablas de restricción, excepto cuando el trigger se dispare como resultado de una restricción ON DELETE CASCADE.

Disparadores de sustitución

Los disparadores de sustitución están asociados a las vistas.

Este tipo de disparadores arrancan al ejecutarse una instrucción de actualización sobre la vista a la que están asociados. Se ejecutan en lugar de (INSTEAD OF) la orden de manipulación que produce el disparo del disparador; por eso se denominan disparadores de sustitución.

El formato genérico para la creación de estos disparadores de sustitución es:

```
CREATE [OR REPLACE] TRIGGER nombretrigger
INSTEAD OF
{DELETE | INSERT | UPDATE [OF <lista_columnas>]}
ON nombrevista
[FOR EACH ROW] [WHEN (condicion)]
<CUERPO DEL TRIGGER (BLOQUE PL/SQL)>
```



Los disparadores de sustitución se ejecutan siempre a nivel de fila.

La cláusula **FOR EACH ROW** es opcional pero no hay otra posibilidad.

Los disparadores de sustitución tienen estas características diferenciales:

- Solamente se utilizan en triggers asociados a vistas, y son especialmente útiles para realizar operaciones de actualización complejas.
- Actúan siempre a nivel de fila, no a nivel de orden, por tanto, a diferencia de lo que ocurre en los disparadores asociados a una tabla, la opción por omisión es FOR EACH ROW.
- No se puede especificar una restricción de disparo mediante la cláusula WHEN (pero no se puede conseguir una funcionalidad similar utilizando estructuras alternativas dentro del bloque PL/SQL).

Uso frecuente de este tipo de disparadores

Sobre una vista podemos hacer un select pero si hacemos un insert, delete o update puede darnos problemas y no dejar ejecutarse la orden correctamente.

Los trigger sobre vistas van a sustituir a estas operaciones por las correspondientes en las tablas que forman la vista.

Un ejemplo de trigger de sustitución seria el siguiente:

```
create or replace trigger t_borrado_emp
    instead of delete on empleado
    for each row
begin
    delete from emple where emp_num=:old.cod
end;
```

Disparadores del sistema

Este tipo de disparadores saltan cuando ocurre un evento del sistema o una instrucción de definición de datos.

Un evento del sistema puede ser el arranque o parada de la base de datos, entrada o salida de un usuario y una instrucción de definición de datos pueden ser la creación, modificación o eliminación de un objeto. La sintaxis para su creación es:

```
CREATE [OR REPLACE]
TRIGGER nombretrigger
{BEFORE | AFTER} {<lista eventos definición> |
<lista eventos del sistema>}
    ON {DATABASE | SCHEMA} [WHEN (condicion)]
<CUERPO DEL TRIGGER
(BLOQUE PL/SQL)>
```

Donde,

- El nombre del trigger puede incluir el esquema mediante la notación de punto.
- <lista eventos definición> puede incluir uno o más eventos DDL separados por OR.
- <lista eventos del sistema> puede incluir uno o más eventos del sistema separados por OR.
- El especificador ON SCHEMA|DATABASE indica el nivel de disparo del trigger:
 - ON DATABASE se disparará siempre que ocurra el evento de disparo.
 - ON SCHEMA se disparará solamente si el evento ocurre en el esquema determinado por el trigger. Por defecto, este esquema es aquel al que pertenece el trigger, pero se puede indicar otro mediante la notación de punto: ON esquema.SCHEMA.

Al asociar un disparador a un evento del sistema debemos indicar el momento del disparo. Algunos eventos sólo pueden disparar ANTES de producirse, otros DESPUÉS y otros admiten las dos posibilidades, tal como se muestra en la tabla siguiente:

Evento	Momento	Se disparan:
STARTUP	AFTER	Después de arrancar la instancia
SHUTDOWN	BEFORE	Antes de apagar la instancia
LOGON	AFTER	Después de que el usuario se conecte a la base de datos.
LOGOFF	BEFORE	Antes de la desconexión de un usuario
SERVERERROR	AFTER	Cuando ocurre un error en el servidor
CREATE	BEFORE AFTER	Antes o después de crear un objeto en el esquema
DROP	BEFORE AFTER	Antes o después de borrar un objeto en el esquema
ALTER	BEFORE AFTER	Antes o después de cambiar un objeto en el esquema
TRUNCATE	BEFORE AFTER	Antes o después de ejecutar un comando truncate
GRANT	BEFORE AFTER	Antes o después de ejecutar un comando grant

Evento	Momento	Se disparan:
REVOKE	BEFORE AFTER	Antes o después de ejecutar un comando revoke
DLL	BEFORE AFTER	Antes o después de ejecutar cualquier comando de definición de datos

PL/SQL dispone de algunas funciones que permiten acceder a los atributos del evento del disparo ORA_SYSEVENT, ORA_LOGIN_USER, ORA_DICT_OBJ_NAME, ORA_DICT_OBJ_TYPE, etcétera. En los manuales del producto podemos encontrar un listado completo de las funciones accesibles desde estos disparadores. Estas funciones pueden usarse en la cláusula WHEN o en el cuerpo del disparador. Cabe mencionar que:

- Desde un disparador LOGON o LOGOFF se puede comprobar el identificador de usuario, o el nombre de usuario (ID, USERID, USERNAME).
- Desde un disparador DDL puede comprobar el tipo y el nombre del objeto que se está modificando (y el ID o nombre de usuario).

Ejemplo:

Escribiremos un disparador que controlará las conexiones de los usuarios en la base de datos. Para ello introducirá en la tabla control_conexiones el nombre de usuario (USER), la fecha y hora en la que se produce el evento de conexión, y la operación CONEXIÓN que realiza el usuario.

Para que el disparador pueda crearse deberá estar creada la tabla control_conexiones:

```
CREATE TABLE control_conexiones (usuario VARCHAR2(20), momento TIMESTAMP,
evento VARCHAR2(20));
```

```

CREATE OR REPLACE TRIGGER ctrl_conexiones
AFTER LOGON
ON DATABASE
BEGIN
    INSERT INTO control_conexiones (usuario, momento, evento)
    VALUES (ORA_LOGIN_USER, SYSTIMESTAMP, ORA_SYSEVENT);
END;

```

Para crear este disparador a nivel ON DATABASE hay que tener el privilegio ADMINISTER DATABASE TRIGGER, de lo contrario sólo nos permitirá crearlo ON SCHEMA.

Una vez creado el disparador cualquier evento de conexión en el esquema producirá el disparo del trigger y la consiguiente inserción de la fila en la tabla.

Por otro parte, crearemos un trigger que inserte en la tabla control_eventos cualquier instrucción de definición de datos:

```

CREATE TABLE control_eventos (usuario VARCHAR2(20), momento TIMESTAMP,
evento VARCHAR2(40));

```

```

CREATE OR REPLACE TRIGGER ctrl_eventos
AFTER DDL
ON DATABASE
BEGIN
    INSERT INTO control_eventos (usuario, momento, evento)
    VALUES (USER, SYSTIMESTAMP, ORA_SYSEVENT || '*' ||
    ORA_DICT_OBJ_NAME);
END;

```

Consideraciones sobre disparadores

Para un buen uso de los disparadores es importante seguir ciertas pautas que nos ayudarán a trabajar mejor con este tipo de elementos.

Consideraciones a tener en cuenta

- Para crear un trigger hay que tener privilegios de CREATE TRIGGER, así como los correspondientes privilegios sobre la tabla o tablas y otros objetos referenciados por el trigger.
- Con el nombre del trigger se puede especificar el esquema en el que queremos crearlo, utilizando la notación de punto. Por omisión, Oracle asumirá nuestro esquema actual en el momento de crear el trigger. El privilegio CREATE ANY TRIGGER permite crear trigger en cualquier esquema.
- La tabla, vista u objeto al que se asocia el disparador puede pertenecer a un esquema distinto del actual, siempre que se tengan los privilegios correspondientes; en este caso, se utilizará la notación de punto. No se puede asociar un trigger a una tabla del esquema SYS.
- Como ya hemos señalado, un trigger forma parte de la operación de actualización que lo disparó y si éste falla, Oracle dará por fallida la actualización. Esta característica puede servir para impedir desde el trigger que se realice una determinada operación, ya que si levantamos una excepción y no la tratamos, el trigger y la operación fallarán.
- Los disparadores son una herramienta muy útil, pero su uso indiscriminado puede degradar el comportamiento de la base de datos y ser fuente de problemas. Por ello, cuando se trata de implementar restricciones de integridad, se deberán utilizar preferentemente las restricciones ya disponibles: PRIMARY KEY, FOREIGN KEY, CHECK, NOT NULL, UNIQUE, CASCADE, SET NULL, SET DEFAULT, etcétera.

Activar y desactivar disparadores

Un disparador puede estar activado o desactivado. Cuando se crea está activado, pero podemos variar esta situación mediante: ALTER TRIGGER nombretrigger DISABLE.

- Para volver a activarlo utilizamos: ALTER TRIGGER nombretrigger ENABLE.
- Para volver a compilar emplearemos: ALTER TRIGGER nombretrigger COMPILE.
- Para eliminar un trigger escribiremos: DROP TRIGGER nombretrigger.



Las vistas `dba_triggers` y `user_triggers` contienen toda la información sobre los trigger.

Buenas prácticas para usar disparadores

- Se debe usar TRIGGER en el caso de eventos específicos que queremos registrar como actividad de la base de datos. **Ejemplo:** si al modificar una fila incluir esos cambios en un log de actividades.
- No hacer TRIGGER para hacer funciones que ya realiza la base de datos. **Ejemplo:** sería la comprobación de datos al incluir en una tabla cuando ya los campos tienen incluidas restricciones.
- No crear TRIGGER que dependan del orden en el que las sentencias de SQL procesan las filas, ya que el orden puede variar. **Ejemplo:** No asignar un valor a una variable de un paquete global en un TRIGGER de filas si el valor actual de la variable depende de la fila que está siendo procesada por el TRIGGER de filas. Si un disparador actualiza variables de paquete globales, se deberán inicializar las variables en las sentencias de un BEFORE TRIGGER.
- Usar BEFORE TRIGGER para filas para modificar la fila antes de que se escriban los datos de la fila en el disco.
- Usar AFTER TRIGGER de filas para obtener el ID de la fila y usarlo en las operaciones.
- No se deben crear TRIGGER recursivos, es decir que se llame a sí mismo
- Si crea un disparador que incluye sentencias que acceden a una base de datos remota, se debe incluir un manejador de excepciones en un subprograma que se lanzará desde el manejador del TRIGGER.

Excepciones

Un disparador si no se ejecuta correctamente puede generar una excepción.

En la mayoría de los casos, si un disparador ejecuta alguna instrucción que genera un error y por lo tanto una excepción, si la excepción no se maneja dentro del disparador con un controlador de excepciones, entonces la base de datos revertirá los efectos tanto de las sentencias ejecutadas por el TRIGGER como la sentencia disparadora que lanzó el evento.

En los siguientes casos, la base de datos revierte sólo los efectos del disparador y no los efectos de la sentencia activadora (además de registrar el error en el archivos de trace y el log de alertas):

- El evento desencadenante es o bien AFTER STARTUP ON DATABASE o BEFORE SHUTDOWN ON DATABASE.
- El evento desencadenante es AFTER LOGON ON DATABASE y el usuario tiene los permisos de ADMINISTER DATABASE TRIGGER.
- El evento desencadenante es AFTER LOGON ON SCHEMA y el usuario, es propietario del esquema o tiene el privilegio ALTER ANY TRIGGER.

Utilización de RAISE_APPLICATION_ERROR

En el paquete DBMS_STANDARD se incluye un procedimiento llamado RAISE_APPLICATION_ERROR que nos sirve para levantar errores y definir mensajes de error personalizados.

Su formato es el siguiente:

```
RAISE_APPLICATION_ERROR(numero_error,mensaje_error);
```

Es importante saber que el número de error está comprendido entre -20000 y -20999 y el mensaje es una cadena de caracteres de hasta 512 bytes.

Este procedimiento crea una excepción que solo puede ser tratada en WHEN OTHERS.

Ejemplos:

En los siguientes ejemplos se muestra cómo se pueden lanzar excepciones una vez que se intentan realizar ciertas acciones con la base de datos o el esquema utilizado.

Ejemplo 1

En este ejemplo se crea un TRIGGER ante un esquema llamado HR. Cuando un usuario se conecta en su intento de borrar un objeto de la base de datos, la base de datos dispara un error indicando que no se puede borrar el objeto.

```
CREATE OR REPLACE TRIGGER triggerBorrado
BEFORE DROP ON hr.SCHEMA
BEGIN
    RAISE_APPLICATION_ERROR
    (
        num => -20000,
        msg => 'No se puede borrar el objeto'
    );
END;
/
```

Ejemplo 2

En el ejemplo se muestra la sintaxis básica de un TRIGGER para registrar los errores. Este TRIGGER se activa después de una ejecución de una sentencia de inicio de sesión sin éxito.

```
CREATE TRIGGER errorLog
  AFTER SERVERERROR ON DATABASE
  BEGIN
    IF (IS_SERVERERROR (1017)) THEN
      (se escribe el código de tratamiento del error de login)
    END IF;
  END;
  /
```

Ejemplo 3

En el ejemplo se ejecuta el TRIGGER usuarioChequeo después de que un usuario inicia sesión en la base de datos.

```
CREATE OR REPLACE TRIGGER usuarioChequeo
  AFTER LOGON ON DATABASE
  BEGIN
    usuarioChequeo;
  EXCEPTION
    WHEN OTHERS THEN
      RAISE_APPLICATION_ERROR
        (-20000, 'Error inesperado: '|| DBMS_Utility.Format_Error_Stack);
  END;
  /
```

Ejercicio resuelto

Empleados

Crear un trigger sobre la tabla empleados para que no se permita que un empleado sea jefe de más de cinco empleados.

```
DROP TABLE empleados;
CREATE TABLE empleados
(dni char(4),
nomemp varchar2(15),
cojefe char(4),
PRIMARY KEY (dni),
FOREIGN KEY (cojefe) references empleados on delete cascade);
```

Para ver el funcionamiento introducimos los siguientes valores en la tabla:

```
INSERT INTO empleados VALUES ('D1','Director',null);
INSERT INTO empleados VALUES ('D2','D.Comercial','D1');
INSERT INTO empleados VALUES ('D3','D.Producción','D1');
INSERT INTO empleados VALUES ('D4','Jefe Ventas','D2');
INSERT INTO empleados VALUES ('D5','Jefe Marketing','D2');
INSERT INTO empleados VALUES ('D6','Vendedor 1','D4');
INSERT INTO empleados VALUES ('D7','Vendedor 2','D4');
INSERT INTO empleados VALUES ('D8','Vendedor 3','D4');
INSERT INTO empleados VALUES ('D9','Vendedor 4','D4');
INSERT INTO empleados VALUES ('D10','Obrero 1','D3');
INSERT INTO empleados VALUES ('D11','Obrero 2','D3');
INSERT INTO empleados VALUES ('D12','Obrero 3','D3');
INSERT INTO empleados VALUES ('D13','Secretaria','D5');
INSERT INTO empleados values ( 'D14','Obrero4','D4');
```

Insertamos valores en la tabla

El trigger para controlar que no haya más de cinco empleados con el mismo jefe sería el siguiente:

```

CREATE OR REPLACE TRIGGER jefes
BEFORE INSERT ON empleados
FOR EACH ROW
DECLARE
    supervisa INTEGER;
BEGIN
    SELECT count(*) INTO supervisa FROM empleados
    WHERE cojefe = :new.cojefe;
    IF supervisa > 4 THEN
        raise_application_error
            (-20600,:new.cojefe||'no se puede supervisar mas de 5');
    END IF;
END;
/

```

Para contar los empleados a los que supervisa cada jefe:

```
SELECT cojefe, count(*) FROM empleados GROUP BY cojefe;
```

Ejemplo de salida:

Una vez que tenemos los datos introducidos si ejecutamos el siguiente script tendríamos la salida que se muestra:

```

INSERT INTO empleados values ( 'D15', 'Obrero5', 'D4');
SELECT COJEFE,COUNT(*) FROM EMPLEADOS GROUP BY COJEFE;

```

Script para añadir un empleado más a D4, que ya tenía cinco trabajadores asociados

```
Salida de Script x
Tarea terminada en 0,01 segundos

Error que empieza en la linea: 18 del comando :
INSERT INTO empleados values ( 'D15','Obrero5','D4')
Informe de error -
Error SQL: ORA-20600: D4 no se puede supervisar mas
de 5
ORA-06512: at "TEL.JEFES", line 8
ORA-04088: error during execution of trigger 'TEL.JEFES'

COJefe COUNT(*)
-----
1
D1 2
D4 5
D5 1
D2 2
D3 3

6 filas seleccionadas
```

