Disparadores

Motivación: Comportamientos Activos

- Las BD tradicionales se consideran pasivas
  - No pueden reaccionar automáticamente ante situaciones
    - Ej.: actualizar automáticamente la ruta del autobús de la escuela cuando se suscriban nuevos alumnos/as a dicho servicio → comprobaciones periódicas
- Encapsular comportamiento en las aplicaciones que acceden a la BD
  - Poca modularidad, comportamiento distribuido
- O en una aplicación de monitorización
  - Más modular, pero ¿con qué frecuencia de monitorización?
- Mejor si podemos encapsularlo en la propia BD → BD activas
  - Mejor modularidad
  - No comprobaciones periódicas → reacción inmediata ante cambios

Ejemplos

- Gestión de inventarios
  - Solicitar un producto al proveedor cuando baje su nivel de existencias por debajo de cierto umbral
- Gestión de viajes
  - Si el autobús está lleno y falta bastante tiempo para el viaje, asignar un autobús adicional
- Estrategias de precios en supermercados
  - Si a un producto perecedero le queda poco para caducar, bajar su precio

Reglas ECA

- Reglas ECA:
  - Evento: qué dispara la acción, suceso a vigilar
  - Condición: situación que debe darse para ejecutar la acción asociada
  - Acción: el comportamiento que se dispara (qué se hace) cuando se produce el evento y la condición es cierta
- Disparador (*trigger*) = procedimiento invocado automáticamente por el SGBD como respuesta a ciertos cambios en la BD (≈ demonio que monitoriza la BD)

Utilización

- Utilización interna para la explotación o administración de la BD:
  - Implementación de restricciones de integridad
  - Actualización de vistas materializadas (si no están soportadas de forma transparente por el SGBD) o atributos derivados
  - Seguridad y auditoría: registro de acciones sobre objetos de la BD
  - Gestión de versiones, mantenimiento de datos históricos
  - Realización automática de copias de los datos
- Utilización externa para la implementación de reglas de negocio:
  - Ej.: si pasan 7 días sin que el cliente haya pagado una factura, se le envía un email, se le bloquea la posibilidad de realizar más compras, y se registra como moroso

Eventos (I)

- En general, la fuente del evento podría ser:
  - Una sentencia del LMD:
    - *Insert*, *Delete*, *Update*
    - Antes o después
  - La finalización de una transacción:
    - *Commit*, *rollback*
  - Una excepción:
    - Violación de permisos de acceso
    - Bloqueos
    - …
  - La aplicación
  - El reloj del sistema

Eventos (II)

- Granularidad de los cambios
  - Disparadores **a nivel de tupla** → un cambio en una sola tupla se considera una ocurrencia del evento
  - Disparadores **a nivel de sentencia** → los cambios en todas las tuplas se consideran una ocurrencia del evento

Condiciones y Acciones

- Condición → un predicado sobre la BD
- Acción:
  - Comandos SQL
  - Comandos PL/SQL (SQL extendido)
  - Llamadas externas
  - Abortar la transacción → *rollback*
    - Ej.: si ningún empleado puede ganar más que su jefe, podemos definir una regla ECA donde el evento sea la inserción en la tabla empleados, la condición que el salario introducido sea mayor que el del jefe, y la acción abortar la transacción

Modos de Acoplamiento

- *Modo de acoplamiento inmediato*:
  - La regla ECA se evalúa tan pronto como se detecta el evento
- *Modo de acoplamiento diferido*:
  - La regla ECA se evalúa al finalizar la transacción donde se ha activado (justo antes de hacer *commit*)
- *Modo de acoplamiento desacoplado*:
  - La evaluación de la condición y la ejecución de la acción se ejecutan en transacciones separadas

Disparadores en Oracle (I)

- Tres tipos en Oracle:
  - DML (*Data Manipulation Language*):
    - Disparados por sentencias DML: *INSERT*, *UPDATE*, *DELETE*
    - Condición adicional con la cláusula *WHEN*
  - *INSTEAD OF*:
    - Definidos sobre vistas
    - Permiten transformar una actualización sobre vistas no directamente actualizables en actualizaciones adecuadas sobre las tablas base
  - *Triggers* sobre eventos de sistema o de usuario:
    - Definidos sobre un esquema (*SCHEMA*) o sobre la BD (*DATABASE*)
    - Eventos de sistema: arranque y parada, transacciones, errores
    - Eventos de usuarios: entrada en el sistema, salida del sistema, sentencias DDL (*Data Definition Language*): *CREATE*, *ALTER*, *DROP*
- Están almacenados en la BD

Disparadores en Oracle (II)

- Estructura:

```
CREATE [OR REPLACE] TRIGGER nombre
{BEFORE | AFTER | INSTEAD OF}
{INSERT | DELETE | UPDATE [OF <atributos>]} ON <tabla>
[REFERENCING  {NEW AS …, OLD AS …}+]
[FOR EACH ROW]
[WHEN condición]
BEGIN
  cuerpo del disparador (bloque PL/SQL)
END;
```

Condiciones → requieren *FOR EACH ROW*; no pueden especificarse para *triggers INSTEAD OF*.
INSTEAD OF ←→ *triggers* sobre vistas.
Warning: Trigger created with compilation errors. → *SHOW ERROR TRIGGER nombreDelTrigger;*
o simplemente *SHOW ERRORS*

Disparadores en Oracle (III)

- Disparadores a nivel de fila (*row triggers*):
  - *FOR EACH ROW*
  - Ejecutan la acción una vez por cada fila que se vea afectada por la sentencia que dispara el *trigger*
  - Lógicamente, no se dispara si no se ve afectada ninguna fila
  - Ejemplo: cuando se borre un cliente de la tabla de clientes, borrarlo también de la tabla de personas
- Disparadores a nivel de sentencia (*statement triggers*):
  - Por defecto (cuando no se indica "*FOR EACH ROW*")
  - Ejecutan una sola vez la acción asociada, con independencia del número de filas afectadas por la sentencia que dispara el *trigger*
  - Ejemplo: cuando se borre en la tabla de clientes mostrar un mensaje de error indicando que no se pueden borrar clientes

Disparadores en Oracle (IV)

- Disparadores *BEFORE*:
  - Ejecutan la acción asociada antes de la ejecución de la sentencia correspondiente
  - Por tanto, pueden utilizarse para decidir si la sentencia debe ejecutarse o no o para cambiar la ejecución de la misma (fijar valores alternativos antes de que se escriban en disco)
- Disparadores *AFTER*:
  - Ejecutan la acción asociada después de la ejecución de la sentencia
- Disparadores *INSTEAD OF*:
  - Ejecutan la acción asociada en lugar de la sentencia correspondiente
  - Muy útil para vistas
- El uso de uno u otro tipo de disparador determina la temporalidad del evento considerado

Disparadores en Oracle (V)

- Una misma sentencia SQL puede disparar varios *triggers*
- Además, la activación de un *trigger* también puede disparar la activación de otros *triggers* (*cascading triggers*)
- Orden de ejecución:
  - *BEFORE statement triggers*
  - *BEFORE row triggers*
  - Ejecución de la sentencia
  - *AFTER row triggers*
  - *AFTER statement triggers*

  Ojo: todas después de ejecutar la sentencia, aunque sea tupla a tupla
- Disparadores del mismo tipo se ejecutan en un orden arbitrario
  - En Oracle 11g, cláusula *FOLLOWS*

Disparadores en Oracle (VI)

- NEW → tupla nueva
- OLD → tupla vieja

  Definidos para disparadores a nivel de fila (se les puede asignar alias con la cláusula REFERENCING)
- De tipo *nombreTabla%ROWTYPE*, donde *nombreTabla* es el nombre de la tabla sobre la que se define el disparador
- *OLD.nombreColumna*:
  - Valor de la columna antes de su cambio por un *UPDATE*
  - Valor de la columna antes de su borrado por un *DELETE*
  - *NULL* en el caso de su inserción por un *INSERT*
- *NEW.nombreColumna*:
  - Valor de la columna después de su cambio por un *UPDATE*
  - Valor de la columna después de su inserción por un *INSERT*
  - *NULL* en el caso de su borrado por un *DELETE*

Disparadores en Oracle (VII)

- Disponibles tanto en disparadores *BEFORE* como *AFTER*:
  - Podemos cambiar valores en *NEW* en el caso de disparadores a nivel de tupla *BEFORE*, pero no en el caso de los AFTER
- Sintaxis:
  - En el cuerpo del disparador → *:NEW*, *:OLD*
  - En la cláusula *WHEN* → *NEW*, *OLD* (sin :)
- Prohibiciones (son pseudo-registros):
  - No se puede cambiar el valor de *OLD*, pero sí el valor de *NEW*
  - No se puede pasar *OLD* o *NEW* como parámetro de tipo registro a un subprograma llamado desde el disparador, pero sí es posible pasar atributos de *OLD* y *NEW*
  - No se pueden realizar operaciones a nivel de registro con *OLD* y *NEW* (p.ej., *:NEW := NULL*)

Disparadores en Oracle (VIII)

```
CREATE OR REPLACE TRIGGER ejemploPredicadosCond
BEFORE INSERT OR UPDATE OR DELETE ON tabla
BEGIN
IF DELETING THEN
  Acciones en caso de borrado
ELSIF INSERTING THEN
  Acciones en caso de inserción
ELSIF UPDATING[('columna1')]
  Acciones en caso de modificación de columna1
ELSIF UPDATING[('columna2')]
  Acciones en caso de modificación de columna2
END IF;
END ejemploPredicadosCond;
/
```

Cuando más de un tipo de operación DML puede disparar el trigger

Para reducir la sobrecarga si no se actualiza la columna de interés (de otro modo, se puede simplemente usar el predicado UPDATING sin indicar la columna)

Disparadores en Oracle (IX)

```
CREATE OR REPLACE TRIGGER ejemploExcepcion
BEFORE INSERT ON tabla
FOR EACH ROW
BEGIN
  IF … THEN
    RAISE_APPLICATION_ERROR(-20001,
      'No se permite insertar porque…');
END IF;
...
END ejemploExcepcion;
/
```

Condición que representa una inserción no permitida

Valor en el rango -20000 .. -20999

Disparadores en Oracle (X)

- Antes del *BEGIN* del disparador:
  - *DECLARE*

    declaración de variables

    TIPO = NUMBER, INTEGER, CHAR(n), DATE, …
- Sintaxis de declaración de variables:
  - nombreVariable TIPO;
  - nombreConstante *CONSTANT* TIPO:=valor;
  - nombreVariable nombreTabla.nombreColumna*%TYPE*
  - nombreVariable nombreTabla*%ROWTYPE*

Disparadores en Oracle (XI)

- En general, cuando se aborta la ejecución de un disparador (ej., con *RAISE_APPLICATION_ERROR*)
  - Se deshacen los efectos tanto del disparador como de la sentencia que activó el disparador
- Alternativamente, es posible definir disparadores que contienen transacciones autónomas (para disparadores y procedimientos almacenados):
  - *DECLARE*

    PRAGMA AUTONOMOUS_TRANSACTION;
  - En este caso tenemos una transacción independiente arrancada desde una *transacción principal*
  - Hace *commit* y *rollback* sin afectar a la transacción principal

Disparadores en Oracle (XII)

```
CREATE TABLE cambiosSalarios(idCambio NUMBER(6), fechaCambio DATE,
    nuevoSalario NUMBER(8,2), viejoSalario NUMBER(8,2));
CREATE OR REPLACE TRIGGER auditarCambioSalario
AFTER UPDATE OF salario ON Empleados
FOR EACH ROW
DECLARE
  PRAGMA AUTONOMOUS_TRANSACTION;
BEGIN
  INSERT INTO cambiosSalarios VALUES(:old.idEmpleado, SYSDATE, :new.salario,
    :old.salario);
  COMMIT;
END;
/
```


Disparadores en Oracle (XIII)

- Tabla mutante: tabla que está siendo modificada por
  - una sentencia SQL (*INSERT*, *UPDATE*, *DELETE*)
  - o por el efecto de un *DELETE CASCADE (*integridad referencial)
- Restricciones sobre tablas mutantes (*ORA-04091* → *rollback*):
  - En el cuerpo de un disparador no se puede consultar o actualizar una tabla mutante (tabla cuya modificación está activando el disparador) → posible excepción: si la sentencia disparadora es un *INSERT* de una única tupla
  - Aplicable a disparadores a nivel de fila
  - Aplicable a disparadores a nivel de sentencia que se disparan como resultado de un borrado en cascada (en principio, sólo hasta Oracle 8i…)
  - Vistas que están siendo modificadas en disparadores *INSTEAD OF* no se consideran mutantes
  - Estas restricciones pretenden evitar que los disparadores puedan ver datos inconsistentes o que se produzca una cascada de ejecución de disparadores

Disparadores en Oracle (XIV)

```
CREATE OR REPLACE TRIGGER ContarEmpleados
AFTER DELETE ON Empleado
FOR EACH ROW
DECLARE
  numEmpleados INTEGER;
BEGIN
  SELECT COUNT(*) INTO numEmpleados FROM Empleado;
  DBMS_OUTPUT.PUT_LINE('Hay ' || numEmpleados || ' empleados.');
END;
```

SET SERVEROUTPUT ON

```
DELETE FROM emp WHERE empno = 7499;
```

→

ORA-04091: table Empleado is mutating, trigger/function might not see it

Disparadores en Oracle (XV)

```
CREATE TABLE Matriculas(
  nombreAlumno VARCHAR2(100),
  nombreAsignatura VARCHAR2(100),
  constraint pk_matricula PRIMARY KEY(nombreAlumno, nombreAsignatura)
);

CREATE OR REPLACE TRIGGER triggerMatriculas
BEFORE INSERT OR UPDATE ON Matriculas
FOR EACH ROW
DECLARE
  numAsignaturas NUMBER;
BEGIN
  SELECT COUNT(*) INTO numAsignaturas FROM Matriculas
  WHERE Matriculas.nombreAlumno = :NEW.nombreAlumno;
  IF numAsignaturas >= 10 THEN
    RAISE_APPLICATION_ERROR (-20000, 'El alumno ' || :NEW.nombreAlumno || ', ya tiene demasiadas asignaturas');
  END IF;
END;
/
```

Disparadores en Oracle (XVI)

```
UPDATE Matriculas
SET nombreAlumno = 'Juancho Rancho'
WHERE nombreAlumno  = 'Lucas White';
```

→

```
UPDATE Matriculas
*
ERROR at line 1:
ORA-04091: table MATRICULAS is mutating, trigger/function may not
    see it
ORA-06512: at "TRIGGERMATRICULAS", line 6
ORA-04088: error during execution of trigger 'TRIGGERMATRICULAS'
```


Disparadores en Oracle (XVII)

- Solución:
  - Creamos un disparador a nivel de fila (*FOR EACH ROW*) para almacenar los datos que queremos consultar
    - Podemos utilizar una tabla auxiliar (o una tabla PL/SQL o un paquete con variables)
  - Creamos un disparador a nivel de sentencia y *AFTER*, donde realizamos la consulta utilizando los datos almacenados previamente

- Otra solución (sólo desde Oracle 11g):

  Disparadores compuestos (*compound triggers*)
  - Se pueden disparar en más de un momento
  - Simplifican la compartición de datos entre las acciones implementadas para esos diversos momentos

Disparadores en Oracle (XVIII)

```
CREATE TABLE tempAlumnos(
  nombreAlumno VARCHAR2(100)
);

CREATE OR REPLACE TRIGGER triggerMatriculas
BEFORE INSERT OR UPDATE ON Matriculas
FOR EACH ROW
DECLARE
  numAsignaturas NUMBER;
BEGIN
IF :NEW.nombreAlumno IS NOT NULL THEN
    BEGIN
      INSERT INTO tempAlumnos VALUES(:NEW.nombreAlumno);
      --dbms_output.put_line('Guardo el alumno: ' || :NEW.nombreAlumno);
    END;
  END IF;
END;
/
```

SET SERVEROUTPUT ON;

Disparadores en Oracle (XIX)

```
CREATE OR REPLACE TRIGGER triggerMatriculasSentencia
AFTER INSERT OR UPDATE ON Matriculas
DECLARE
  numAsignaturas NUMBER;
  nombreAlumnoLocal Matriculas.nombreAlumno%TYPE;
  CURSOR cursorNombresAlumnos IS SELECT nombreAlumno FROM tempAlumnos;
BEGIN
OPEN cursorNombresAlumnos;
  LOOP
    FETCH cursorNombresAlumnos INTO nombreAlumnoLocal;
    EXIT WHEN cursorNombresAlumnos%NOTFOUND;
    SELECT COUNT(*) INTO numAsignaturas FROM Matriculas WHERE Matriculas.nombreAlumno = nombreAlumnoLocal;
    IF numAsignaturas >= 10 THEN
      RAISE_APPLICATION_ERROR (-20000, 'El alumno ' || nombreAlumnoLocal || ', ya tiene demasiadas asignaturas');
    END IF;
  END LOOP;
  CLOSE cursorNombresAlumnos;
  DELETE FROM tempAlumnos;
END;
/
```

Disparadores en Oracle (XX)

- Si intentamos una actualización que hace que el número de matrículas de un estudiante pase de 9:

```
UPDATE Matriculas
SET nombreAlumno = 'Juancho Rancho'
WHERE nombreAlumno  = 'Lucas White';
→
UPDATE Matriculas
    *
ERROR at line 1:
ORA-20000: El alumno Juancho Rancho, ya tiene demasiadas asignaturas
ORA-06512: at "TRIGGERMATRICULASSENTENCIA", line 13
ORA-04088: error during execution of trigger 'TRIGGERMATRICULASSENTENCIA'
```

Se hace *rollback* de la transacción

Disparadores en Oracle (XXI)

Añade al proyecto las horas trabajadas por un empleado

```
CREATE or REPLACE TRIGGER ContabHorasProy
AFTER UPDATE ON Participar
FOR EACH ROW
WHEN (NEW.numHoras > 0)
 begin
   UPDATE Proyecto
   SET horas = horas + :NEW.numHoras - :OLD.numHoras
   WHERE  numProy = :NEW.numProy;
 end ContabHorasProy;
/
```

Fuente: transparencias de Santiago Velilla

Disparadores en Oracle (XXII)

Refleja en HistorialSalario el cambio de sueldo de un empleado

```
CREATE OR REPLACE TRIGGER GuardarHistorialSalario
BEFORE UPDATE ON Empleado
FOR EACH ROW
 begin
  if (:OLD.Salario <> :NEW.salario)
   then INSERT INTO HistorialSalario VALUES (:OLD.codEmp,
     :OLD.salario, sysdate);
  end if;
 end GuardarHistorialSalario;
/
```

Fuente: transparencias de Santiago Velilla

Disparadores en Oracle (XXIII)

Problemas con la actualización de vistas (1)

```
CREATE TABLE Persona(
  nombre VARCHAR2(20),
  descripcion VARCHAR2(20),
  profesion VARCHAR(15),
  constraint pk_persona PRIMARY KEY(nombre)
);

CREATE VIEW Estudiante AS
  (SELECT nombre, descripcion FROM Persona WHERE
    profesion='Estudiante');
```

Disparadores en Oracle (XXIV)

Problemas con la actualización de vistas (2)

```
INSERT INTO Estudiante VALUES('Juan', 'Muy trabajador...');
```

Curiosamente, si ahora consultamos los estudiantes:

```
SELECT * FROM Estudiante;
```

no sale Juan, ya que se adoptó un valor *NULL* para *"profesion"* al insertarlo en *Persona*

Disparadores en Oracle (XXV)

Problemas con la actualización de vistas (3): solución

```
CREATE OR REPLACE TRIGGER triggerEstudiantes
INSTEAD OF INSERT ON Estudiante
FOR EACH ROW
BEGIN
  INSERT INTO Persona VALUES(:NEW.nombre, :NEW.descripcion, 'Estudiante');
END;
/
```

La actualización de algunas vistas sólo es posible mediante *triggers INSTEAD OF*

BD de Piezas: Esquema

```
CREATE TABLE Pieza     (
 clvPieza  NUMBER(9) PRIMARY KEY,
 nombPieza CHAR(32) NOT NULL,
 color     CHAR(32));

CREATE TABLE Proveedor  (
 clvProv  NUMBER(9) PRIMARY KEY,
 nombProv CHAR(32) NOT NULL);

CREATE TABLE suministrar (
 clvProv  NUMBER(9),
 clvPieza NUMBER(9),
 PRIMARY KEY (clvProv, clvPieza),
 FOREIGN KEY (clvProv)  REFERENCES Proveedor(clvProv),
 FOREIGN KEY (clvPieza) REFERENCES Pieza(clvPieza) ON DELETE cascade);
```

Santiago Velilla (adaptado de)

BD de Piezas: Ej. Disparador Histórico Operaciones (I)

- Crear la tabla para el histórico de operaciones en Proveedor:

```
CREATE TABLE hist_Proveedor (
  idOp number(5) CONSTRAINT idOp_PK PRIMARY KEY,
  clvProv number(9) NOT NULL,
  nombProv char(32) NOT NULL,
  fecha date NOT NULL,
  empleado varchar2(255) NOT NULL,
  tpOp char(2) NOT NULL);
```

Santiago Velilla (adaptado de)

BD de Piezas: Ej. Disparador Histórico Operaciones (II)

- Crear una secuencia para generar el idOp de las operaciones:

CREATE SEQUENCE idHistOpSQ;

BD de Piezas: Ej. Disparador Histórico Operaciones (III)

```
CREATE OR REPLACE TRIGGER histProveedor_TR
AFTER INSERT OR UPDATE OR DELETE ON Proveedor
FOR EACH ROW
  DECLARE
    idProv Proveedor.clvProv%TYPE;
    nombre Proveedor.nombProv%TYPE;
    Operac hist_Proveedor.tpOp%TYPE;
BEGIN
…
END histPiezas_TR;
/
```

Santiago Velilla (adaptado de)

BD de Piezas: Ej. Disparador Histórico Operaciones (IV)

```
BEGIN
 IF INSERTING THEN Operac := 'I'; idProv := :NEW.clvProv; nombre := :NEW.nombProv;
   ELSIF DELETING THEN
      Operac := 'D'; idProv := :OLD.clvProv;
      nombre := :OLD.nombProv;
   ELSIF UPDATING THEN
      Operac := 'U';
      idProv := :NEW.clvProv;
      nombre := :NEW.nombProv;
  END IF;
  INSERT INTO hist_Proveedor VALUES (idHistOpSQ.NEXTVAL, idProv, nombre, SYSDATE,
USER, Operac);
END histPiezas_TR;
/
```

Santiago Velilla (adaptado de)

BD de Piezas: Ej. Disparador Histórico Operaciones (V)

- Comprobar si hay errores:

  SHOW ERRORS TRIGGER histProveedor_TR;

- *Probar el disparador:*

```
INSERT INTO Proveedor VALUES (1, 'PEREZ');
UPDATE Proveedor SET clvProv=2, nombProv='LUIS'
  WHERE    clvProv=1;
INSERT INTO Proveedor VALUES (1, 'PEREZ');
INSERT INTO Proveedor VALUES (3, 'MARIANO');
DELETE FROM Proveedor WHERE clvProv=1;
```

Santiago Velilla (adaptado de)

BD de Piezas: Ej. Disparador Histórico Operaciones (VI)

- Formato de la salida:
  - – Número máximo de caracteres por línea:

    SET LINESIZE 140
  - – Número de líneas por página (0: quita los saltos y también el encabezamiento):

    SET pagesize 200
  - – Formatos de columnas:

    column empleado format A20

    column nombProv format A32

    column idOp format 999

    A: alphabetic

    http://docs.oracle.com/cd/B19306_01/server.102/b14357/ch12013.htm

BD de Piezas: Ej. Disparador Histórico Operaciones (VII)

- Comprobación de los registros realizados:

  *SELECT * FROM hist_Proveedor;*

```
SQL> SELECT * FROM hist_Proveedor;

IDOP    CLVPROV NOMBPROV                         FECHA     EMPLEADO             TP
---- ---------- -------------------------------- --------- -------------------- --
   1          1 PEREZ                            15-MAY-14 SILARRI              I
   2          2 LUIS                             15-MAY-14 SILARRI              U
   3          1 PEREZ                            15-MAY-14 SILARRI              I
   4          3 MARIANO                          15-MAY-14 SILARRI              I
   5          1 PEREZ                            15-MAY-14 SILARRI              D
```


BD de Piezas: Disparador Máximo Proveedores por Pieza = 3 (I)

```
CREATE OR REPLACE TRIGGER tst_maxProv_TR
BEFORE INSERT ON Suministrar
FOR EACH ROW
DECLARE
  totProv NUMBER;
BEGIN
…
END tst_maxProv_TR;
/
```

Santiago Velilla (adaptado de)

BD de Piezas: Disparador Máximo Proveedores por Pieza = 3 (II)

```
BEGIN
  SELECT count(*) INTO totProv FROM Suministrar
    WHERE :NEW.clvPieza = clvPieza;
  DBMS_OUTPUT.PUT_LINE('total proveedores de pieza '||'
    '||totProv);
  IF totProv > 2 THEN
    raise_application_error( -20501, 'max. proveedores para la pieza!');
  END IF;
END tst_maxProv_TR;
/
```

Santiago Velilla (adaptado de)

BD de Piezas: Disparador Máximo Proveedores por Pieza = 3 (III)

- Probando el disparador:

INSERT INTO Proveedor VALUES (2, 'LOPEZ');

INSERT INTO Pieza VALUES (95, 'TUERCA', 'AZUL');

INSERT INTO suministrar VALUES (2, 95);

BD de Piezas: Disparador Máximo Proveedores por Pieza = 3 (IV)

- Seguimos probando el disparador:

INSERT INTO Suministrar SELECT 2, clvPieza FROM Suministrar WHERE clvPieza=95;

```
SQL> INSERT INTO suministrar SELECT 2, clvPieza FROM Suministrar WHERE clvPieza=95;
INSERT INTO suministrar SELECT 2, clvPieza FROM Suministrar WHERE clvPieza=95
            *
ERROR at line 1:
ORA-04091: table SILARRI.SUMINISTRAR is mutating, trigger/function may not see it
ORA-06512: at "SILARRI.TST_MAXPROV_TR", line 4
ORA-04088: error during execution of trigger 'SILARRI.TST_MAXPROV_TR'
```

En el caso concreto de *"INSERT INTO suministrar VALUES (2, 95);"* no se considera mutante

BD de Piezas: Disparador Máximo Proveedores por Pieza = 3 (V)

- ¿Y si hubiéramos planteado un *disparador AFTER*?

```
CREATE OR REPLACE TRIGGER tst_maxProv_TR AFTER INSERT ON Suministrar
FOR EACH ROW
DECLARE
  totProv NUMBER;
BEGIN
  SELECT count(*) INTO totProv FROM Suministrar WHERE :NEW.clvPieza = clvPieza;
  DBMS_OUTPUT.PUT_LINE('total proveedores de pieza '||' '||totProv);
  IF totProv > 3 THEN raise_application_error( -20501, 'max. proveedores para la pieza!');
END IF;
END tst_maxProv_TR;
/
```


BD de Piezas: Disparador Máximo Proveedores por Pieza = 3 (VI)

- Probando el nuevo disparador:

DELETE FROM Suministrar;

INSERT INTO Suministrar VALUES (2, 95);

```
SQL> delete from suministrar;

1 row deleted.

SQL> INSERT INTO suministrar VALUES (2, 95);
INSERT INTO suministrar VALUES (2, 95)
*
ERROR at line 1:
ORA-04091: table SILARRI.SUMINISTRAR is mutating, trigger/function may not see it
ORA-06512: at "SILARRI.TST_MAXPROV_TR", line 4
ORA-04088: error during execution of trigger 'SILARRI.TST_MAXPROV_TR'
```

BD de Piezas: Disparador Máximo Proveedores por Pieza = 3 (VII)

- Añadamos 1 atributo para mantener el número de proveedores de cada pieza:

ALTER TABLE Pieza ADD(numProv NUMBER);

- Actualicemos su valor:

```
CREATE OR REPLACE PROCEDURE actualizarSuministros_PR
IS BEGIN
  UPDATE Pieza P SET numProv=(SELECT count(*) FROM Suministrar WHERE
clvPieza = P.clvPieza);
END actualizarSuministros_PR;
/
EXECUTE actualizarSuministros_PR
```

Santiago Velilla (adaptado de)

BD de Piezas: Disparador Máximo Proveedores por Pieza = 3 (VIII)

- Y ahora redefinamos el disparador:

```
CREATE OR REPLACE TRIGGER tst_maxProv_TR BEFORE INSERT ON Suministrar
FOR EACH ROW
DECLARE
  totProv NUMBER;
BEGIN
  SELECT numProv INTO totProv FROM Pieza WHERE :NEW.clvPieza = clvPieza;
  DBMS_OUTPUT.PUT_LINE('total proveedores de pieza '||' '||totProv);
  IF totProv > 2 THEN raise_application_error( -20501, 'max. proveedores para la pieza!');
  ELSE UPDATE Pieza SET numProv=numProv+1 WHERE clvPieza = :NEW.clvPieza;
END IF;
END tst_maxProv_TR;
/
```

Santiago Velilla (adaptado de)

BD de Piezas: Disparador Máximo Proveedores por Pieza = 3 (IX)

- Probando este nuevo disparador:

```
DELETE FROM Suministrar;
DELETE FROM Proveedor;
DELETE FROM Pieza;
INSERT INTO Proveedor VALUES (1,'PEREZ');
INSERT INTO Proveedor VALUES (2, 'LOPEZ');
INSERT INTO Proveedor VALUES (3, 'MARTINEZ');
INSERT INTO Proveedor VALUES (4, 'ARNANA');
INSERT INTO Pieza VALUES (91, 'TUERCA',   'ROJO', 0);
INSERT INTO Pieza VALUES (92, 'TUERCA',   'VERDE', 0);
```


BD de Piezas: Disparador Máximo Proveedores por Pieza = 3 (X)

```
INSERT INTO suministrar VALUES (2, 91);
INSERT INTO suministrar VALUES (3, 91);
INSERT INTO suministrar VALUES (1, 91);
INSERT INTO suministrar VALUES (4, 91);
```

```
SQL> INSERT INTO suministrar VALUES (2, 91);

1 row created.

SQL> INSERT INTO suministrar VALUES (3, 92);

1 row created.

SQL> INSERT INTO suministrar VALUES (3, 91);

1 row created.

SQL> INSERT INTO suministrar VALUES (1, 91);

1 row created.

SQL> INSERT INTO Proveedor VALUES (4, 'ARNANA');

1 row created.

SQL> INSERT INTO suministrar VALUES (4, 91);
INSERT INTO suministrar VALUES (4, 91)
            *
ERROR at line 1:
ORA-20501: max. proveedores para la pieza!
ORA-06512: at "SILARRI.TST_MAXPROV_TR", line 6
ORA-04088: error during execution of trigger 'SILARRI.TST_MAXPROV_TR'
```

BD de Piezas: Disparador Máximo Proveedores por Pieza = 3 (XI)

- Veamos otra solución sin atributo derivado y evitando problemas de tablas mutantes:
- Primero creamos una tabla temporal:

```
DROP TABLE tempSum;
CREATE TABLE tempSum (
  clvProv number(9), clvPieza number(9)
);
```


BD de Piezas: Disparador Máximo Proveedores por Pieza = 3 (XII)

- Después un disparador *BEFORE*:

```
CREATE OR REPLACE TRIGGER tst_maxProv_TR
BEFORE INSERT ON Suministrar
FOR EACH ROW
BEGIN
  INSERT INTO tempSum VALUES(:NEW.clvProv, :NEW.clvPieza);
  DBMS_OUTPUT.PUT_LINE('intento de incorporar suministro ' ||
    :NEW.clvProv || ' ' || :NEW.clvPieza);
END tst_maxProv_TR;
/
```


BD de Piezas: Disparador Máximo Proveedores por Pieza = 3 (XIII)

- Y finalmente un disparador *AFTER*:

```
CREATE OR REPLACE TRIGGER nuevosSuministros_TR
AFTER INSERT OR UPDATE ON Suministrar
DECLARE
  laPieza Suministrar.clvPieza%TYPE;
  elProveedor Suministrar.clvProv%TYPE;
  totalProv number;
  CURSOR selCambio IS SELECT clvPieza, clvProv FROM tempSum;
BEGIN
...
```


BD de Piezas: Disparador Máximo Proveedores por Pieza = 3 (XIV)

```
...
  OPEN selCambio;
  LOOP FETCH selCambio INTO laPieza, elProveedor;
  EXIT WHEN selCambio%NOTFOUND;
  SELECT count(*) INTO totalProv FROM Suministrar S
    WHERE S.clvPieza = laPieza;
...
```

BD de Piezas: Disparador Máximo Proveedores por Pieza = 3 (XV)

```
…
    IF totalProv > 3 THEN DBMS_OUTPUT.PUT_LINE('la pieza '||
      laPieza || ' ya tiene '|| (totalProv-1) || ' proveedores');
      raise_application_error( -20501, 'num. max. de '||
        'proveedores alcanzado');
    END IF;
  END LOOP;
  DELETE FROM tempSum;
END tst_maxProv_TR;
/
```

Santiago Velilla (adaptado de)

BD de Piezas: Disparador Máximo Proveedores por Pieza = 3 (XVI)

- Probemos el disparador:

```
DELETE FROM Suministrar;
DELETE FROM Proveedor;
DELETE FROM Pieza;
INSERT INTO Proveedor VALUES (1,'PEREZ');
INSERT INTO Proveedor VALUES (2, 'LOPEZ');
INSERT INTO Proveedor VALUES (3, 'MARTINEZ');
INSERT INTO Proveedor VALUES (4, 'ARNANA');
INSERT INTO Pieza VALUES (91, 'TUERCA',   'ROJO');
INSERT INTO Pieza VALUES (92, 'TUERCA',   'VERDE');
```


BD de Piezas: Disparador Máximo Proveedores por Pieza = 3 (XVII)

```
INSERT INTO Suministrar VALUES (2, 91);
INSERT INTO Suministrar VALUES (3, 91);
INSERT INTO Suministrar VALUES (1, 91);
INSERT INTO Suministrar VALUES (4, 91);
```

```
SQL> INSERT INTO suministrar VALUES (2, 91);
intento de incorporar suministro 2 91

1 row created.

SQL> INSERT INTO suministrar VALUES (3, 91);
intento de incorporar suministro 3 91

1 row created.

SQL> INSERT INTO suministrar VALUES (1, 91);
intento de incorporar suministro 1 91

1 row created.

SQL> INSERT INTO suministrar VALUES (4, 91);
intento de incorporar suministro 4 91
la pieza 91 ya tiene 3 proveedores
INSERT INTO suministrar VALUES (4, 91)
            *
ERROR at line 1:
ORA-20501: num. max de proveedores alcanzado
ORA-06512: at "SILARRI.NUEVOSSUMINISTROS_TR", line 13
ORA-04088: error during execution of trigger 'SILARRI.NUEVOSSUMINISTROS_TR'
```


BD de Piezas: Disparador Máximo Proveedores por Pieza = 3 (XVIII)

- Pero cuidado (el disparador es para INSERT…):

```
INSERT INTO Suministrar VALUES (4, 92);

UPDATE Suministrar
SET clvPieza=91
WHERE clvPieza=92;
```

```
SQL> INSERT INTO Suministrar VALUES (4, 92);
intento de incorporar suministro 4 92

1 row created.

SQL> UPDATE Suministrar
  2  SET clvPieza=91
  3  WHERE clvPieza=92;

1 row updated.
```

```
SQL> select COUNT(clvProv) FROM Suministrar WHERE clvPieza=91;

COUNT(CLVPROV)
--------------
             4

SQL>
```

BD de Piezas: Detectar Creación y Eliminación de Elementos Esquema (I)

```
CREATE OR REPLACE TRIGGER CambiosEsq
AFTER CREATE OR DROP ON SCHEMA
BEGIN
  IF ORA_SYSEVENT = 'CREATE' THEN
    DBMS_OUTPUT.PUT_LINE(ORA_DICT_OBJ_TYPE || ' de nombre ' ||
ORA_DICT_OBJ_NAME || ' se ha creado');
  ELSIF ORA_SYSEVENT = 'DROP' THEN
    DBMS_OUTPUT.PUT_LINE(ORA_DICT_OBJ_TYPE || ' de nombre ' ||
ORA_DICT_OBJ_NAME || ' se ha eliminado' );
  END IF;
END;
/
```

Santiago Velilla (adaptado de)

BD de Piezas: Detectar Creación y Eliminación de Elementos Esquema (II)

- Probando el disparador:

```
CREATE TABLE tempSum (
  clvProv number(9), clvPieza number(9)
);
DROP TABLE tempSum;
```

```
SQL> CREATE TABLE tempSum (
  2     clvProv number(9), clvPieza number(9)
  3  );
TABLE de nombre TEMPSUM se ha creado

Table created.

SQL> DROP TABLE tempSum;
TABLE de nombre TEMPSUM se ha eliminado

Table dropped.
```

Santiago Velilla (adaptado de)

BD de Piezas: Impedir Modificación Nombre de Piezas (I)

```
CREATE OR REPLACE TRIGGER updateNombPieza_TR
BEFORE UPDATE ON Pieza
FOR EACH ROW
DECLARE
 laPieza number;
BEGIN
 laPieza := :OLD.clvPieza;
 IF UPDATING('NOMBPIEZA') THEN
  DBMS_OUTPUT.PUT_LINE('no se permite cambiar el nombre de una pieza');
  raise_application_error(-20501, 'error al modificar la pieza '|| :OLD.clvPieza);
 END IF;
END updateNombPieza_TR;
/
```

Santiago Velilla (adaptado de)

BD de Piezas: Impedir Modificación Nombre de Piezas (II)

- Probando el disparador (con la definición original de *Pieza*):

```
INSERT INTO Pieza VALUES(1, 'TUERCA', 'ROJA');
UPDATE Pieza SET color='ROJIZA' WHERE color='ROJA';
UPDATE Pieza SET nombPieza='TORNILLO';
```

```
SQL> UPDATE Pieza SET nombPieza='TORNILLO';
no se permite cambiar el nombre de una pieza
UPDATE Pieza SET nombPieza='TORNILLO'
       *
ERROR at line 1:
ORA-20501: error al modificar la pieza 1
ORA-06512: at "SILARRI.UPDATENOMBPIEZA_TR", line 7
ORA-04088: error during execution of trigger 'SILARRI.UPDATENOMBPIEZA_TR'
```

Oracle: Gestión de Disparadores (I)

- *SELECT TRIGGER_NAME*

FROM DBA_TRIGGERS;

- *SELECT TRIGGER_NAME*

FROM USER_TRIGGERS;

- *SELECT TRIGGER_NAME*

  FROM USER_TRIGGERS

  WHERE TABLE_NAME='Suministrar';

Oracle: Gestión de Disparadores (II)

- *SELECT TRIGGER_NAME, TRIGGER_BODY*

FROM USER_TRIGGERS

WHERE TRIGGER_NAME = 'NUEVOSSUMINISTROS_TR';

Tiene que ir en mayúsculas

- *TRIGGER_BODY* es de tipo LONG → por defecto, Oracle sólo mostrará 80 caracteres:
  - *SET LONG 2000* (por ejemplo)

Oracle: Gestión de Disparadores (III)

- Vistas del diccionario de datos con información sobre disparadores: *USER_TRIGGERS* , *ALL_TRIGGERS*, *DBA_TRIGGERS*
- Algunos atributos interesantes:
  - *TRIGGER_NAME*: nombre del disparador
  - *TRIGGER_TYPE*: *BEFORE STATEMENT*, *BEFORE EACH ROW*, *BEFORE EVENT*, *AFTER STATEMENT*, *AFTER EACH ROW*, *AFTER EVENT*
  - *BASE_OBJECT_TYPE*: *TABLE*, *VIEW*, *DATABASE*, *SCHEMA*
  - *TABLE_NAME* (nulo si el objeto base no es *TABLE* o *VIEW*)
  - *TRIGGERING_EVENT*: evento que dispara el *trigger*
  - *TRIGGER_BODY*: cuerpo del disparador
  - *WHEN_CLAUSE*: cláusula WHEN definida para el disparador
  - *STATUS*: *ENABLED*, *DISABLED*
  - *DESCRIPTION*: descripción del disparador (útil para recrearlo)

Oracle: Gestión de Disparadores (IV)

- Al crearse, los disparadores quedan habilitados por defecto
- *ALTER TRIGGER nuevosSuministros_TR DISABLE;*
- *ALTER TRIGGER nuevosSuministros_TR ENABLE;*
- *ALTER TABLE Suministrar ENABLE ALL TRIGGERS;*
- *ALTER TABLE Suministrar DISABLE ALL TRIGGERS;*
- *SELECT TRIGGER_NAME, STATUS*

FROM USER_TRIGGERS

WHERE TRIGGER_NAME = 'NUEVOSSUMINISTROS_TR';

Oracle: Gestión de Disparadores (V)

- Vistas del diccionario de datos con información sobre disparadores: *USER_TRIGGERS* , *ALL_TRIGGERS*, *DBA_TRIGGERS*
- Algunos atributos interesantes:
  - *TRIGGER_NAME*: nombre del disparador
  - *TRIGGER_TYPE*: *BEFORE STATEMENT*, *BEFORE EACH ROW*, *BEFORE EVENT*, *AFTER STATEMENT*, *AFTER EACH ROW*, *AFTER EVENT*
  - *BASE_OBJECT_TYPE*: *TABLE*, *VIEW*, *DATABASE*, *SCHEMA*
  - *TABLE_NAME* (nulo si el objeto base no es *TABLE* o *VIEW*)
  - *TRIGGERING_EVENT*: evento que dispara el *trigger*
  - *TRIGGER_BODY*: cuerpo del disparador
  - *WHEN_CLAUSE*: cláusula WHEN definida para el disparador
  - *STATUS*: *ENABLED*, *DISABLED*
  - *DESCRIPTION*: descripción del disparador (útil para recrearlo)

Oracle: Gestión de Disparadores (VI)

Algunos ejemplos

```
SELECT TRIGGER_NAME, STATUS FROM
USER_TRIGGERS;

SELECT TRIGGER_BODY
FROM USER_TRIGGERS
WHERE TRIGGER_NAME='nombreDeMiDisparador';

SELECT TRIGGER_TYPE, TRIGGERING_EVENT, TABLE_NAME
FROM USER_TRIGGERS
WHERE TRIGGER_NAME = 'nombreDeMiDisparador ';
```


Oracle: Gestión de Disparadores (VII)

- *DROP TRIGGER histProveedor_TR;*
- *DROP TRIGGER nuevosSuministros_TR;*
- *DROP TRIGGER tst_maxProv_TR;*
- *DROP TRIGGER updateNombPieza_TR;*
- *DROP TRIGGER CambiosEsq;*