



*Administración de
Bases de Datos
(Grado en Ingeniería Informática)*

Tema 5. Disparadores y Trabajos

Disparadores: *TRIGGER*

- ▶ Bloque PL/SQL → se Ejecuta de forma Implícita
 - Con Operación DML: INSERT, DELETE o UPDATE
 - Con Operación DDL o evento de la Base de datos.
 - Contrariamente, los procedimientos y las funciones se ejecutan haciendo una llamada Explícita a ellos.
 - Un Disparador No admite Argumentos.
- ▶ Utilidad: Sus aplicaciones son inmensas, como por ejemplo:
 - Mantenimiento de **Restricciones** de Integridad complejas.
 - Ej: Restricciones de Estado (como que el sueldo sólo puede aumentar).
 - Auditoría de una Tabla, registrando los cambios efectuados y la identidad del que los llevó a cabo.
 - Lanzar cualquier acción cuando una tabla es modificada.

Disparadores: *TRIGGER*

► Formato:

```
CREATE [OR REPLACE] TRIGGER <NombreT>
{BEFORE|AFTER} <Suceso_Disparo> ON <Tabla>
[FOR EACH ROW [WHEN <Condición_Disparo>]]
    <Bloque_del_TRIGGER>;
```

- <Suceso_Disparo> es la operación DML que efectuada sobre <Tabla> disparará el *trigger*. INSERT, DELETE o UPDATE
 - Puede haber varios sucesos separados por la palabra OR.

Elementos de un TRIGGER

- ▶ **Nombre:** Se recomienda que identifique su función y la tabla sobre la que se define.
- ▶ **Tipos de Disparadores:** Hay 12 Tipos básicos según:
 - **Orden, Suceso u Operación DML:** INSERT, DELETE o UPDATE.
 - Si la orden `UPDATE` lleva una lista de atributos el Disparador sólo se ejecutará si se actualiza alguno de ellos: `UPDATE OF <Lista_Atributos>`
 - **Temporización:** BEFORE (anterior) o AFTER (posterior).
 - Define si el disparador se activa antes o después de que se ejecute la operación DML causante del disparo.
 - **Nivel:** a Nivel de Orden o a Nivel de Fila (FOR EACH ROW).
 - **Nivel de Orden** (*statement trigger*): Se activan sólo una vez, antes o después de la Orden u operación DML.
 - **Nivel de Fila** (*row trigger*): Se activan una vez por cada Fila afectada por la operación DML (una misma operación DML puede afectar a varias filas).

Elementos de un TRIGGER

- ▶ **Ejemplo:** Guardar en una tabla de control la fecha y el usuario que modificó la tabla Empleados:
- ▶ (NOTA: Este trigger es **AFTER** y a nivel de orden)

```
CREATE OR REPLACE TRIGGER Control_Empleados
  AFTER INSERT OR DELETE OR UPDATE ON Empleados
BEGIN
  INSERT INTO Ctrl_Empleados (Tabla, Usuario, Fecha)
    VALUES ('Empleados', USER, SYSDATE);
END Control_Empleados;
```

Disparadores: Ejemplos

- **Ejemplo:** Controlar que la tabla de empleados no sufre ninguna variación durante los fines de semana ni fuera del horario laboral:

```
-- Generar Error si es fuera del horario laboral
CREATE TRIGGER horario_de_cambios
  BEFORE DELETE OR INSERT OR UPDATE ON empleados
BEGIN /* Si es sábado o domingo devuelve un error.*/
  IF (TO_CHAR(SYSDATE, 'DY')='SÁB' OR TO_CHAR(SYSDATE, 'DY')='DOM')
  THEN raise_application_error( -20501,
    'No cambiar datos de empleados durante el fin de semana');
  END IF;
  /* Si hora es anterior a las 8:00 o posterior a las 18:00 */
  IF (TO_CHAR(SYSDATE, 'HH24')<8 OR TO_CHAR(SYSDATE, 'HH24')>=18)
  THEN raise_application_error( -20502,
    'No modificar estos datos fuera del horario de la empresa');
  END IF;
END horario_de_cambios;
```

RAISE_APPLICATION_ERROR permite que un programa PL/SQL pueda **Generar Errores** tal y como lo hace Oracle: El primer argumento es un número entre -20.000 y -20.999. El segundo es el mensaje.

Orden en la Ejecución de Disparadores

- ▶ Una tabla puede tener distintos Tipos de **Disparadores** asociados a una misma **orden DML**.
- ▶ En tal caso, el **Algoritmo de Ejecución** es:
 - 1. Ejecutar, si existe, el disparador tipo **BEFORE** a nivel de orden.
 - 2. Para cada fila a la que afecte la orden: (esto es como un bucle, para cada fila)
 - a) Ejecutar, si existe, el disparador **BEFORE** a nivel de fila, sólo si dicha fila cumple la condición de la cláusula **WHEN** (si existe).
 - b) Ejecutar la propia orden.
 - c) Ejecutar, si existe, el disparador **AFTER** a nivel de fila, sólo si dicha fila cumple la condición de la cláusula **WHEN** (si existe).
 - 3. Ejecutar, si existe, el disparador tipo **AFTER** a nivel de orden.

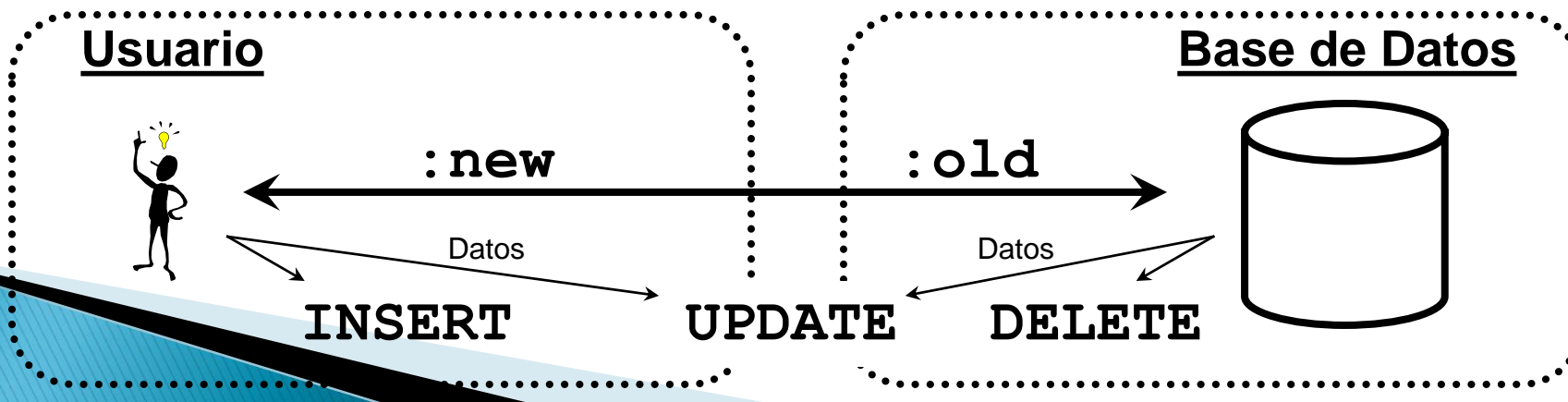
Disparadores de Fila: :old y :new

► Disparadores a Nivel de Fila:

- Se ejecutan una vez por cada Fila procesada por la orden DML disparadora.
- Para acceder a la Fila Procesada: se usan dos Pseudo-Registros de tipo <TablaDisparo>%ROWTYPE: :old y :new

Orden DML	:old	:new
INSERT	No Definido: NULL \forall campo.	Valores Nuevos a Insertar.
UPDATE	Valores Originales (antes de la orden).	Valores Actualizados (después).
DELETE	Valores Originales (antes de la orden).	No Definido: NULL \forall campo.

◦ Esquema:



Disparadores a Nivel de Fila: Ejemplo

- **Ejemplo:** Programar un Disparador que calcule el campo código de Pieza (Codigo) cada vez que se inserte una nueva pieza:

⚠️: No rellena huecos, si existen.

```
CREATE OR REPLACE TRIGGER NuevaPieza
  BEFORE INSERT ON Pieza FOR EACH ROW
BEGIN
  -- Establecer el nuevo número de pieza:
  SELECT MAX(Codigo)+1 INTO :new.Codigo FROM Pieza;
  IF :new.Codigo IS NULL THEN
    :new.Codigo := 1;
  END IF;
END NuevaPieza;
```

- Cuando se ejecute la orden **DML** se emplean los valores del **Pseudo-Registro :new**. Una orden válida sería la siguiente, sin que produzca un error por no tener la clave:
INSERT INTO Pieza (Nombre, Peso) VALUES ('Alcayata', 0.5);
 - En este caso, si se suministra un valor para la clave primaria, tampoco producirá un error, pero ese valor será ignorado y sustituido por el nuevo valor calculado por el disparador.

Disparadores a Nivel de Fila: Ejemplo

- Modificar Pseudo-Registros:
 - No puede modificarse el Pseudo-Registro **:new** en un disparador **AFTER** a nivel de fila.
 - El Pseudo-Registro **:old** nunca se modificará: Sólo se leerá.
- Otro Ejemplo: Si se actualiza el código de una Pieza, actualizar el código de los Suministros en los que se usaba:
-- Actualizar Suministros.Pieza si cambia Pieza.Codigo:

```
CREATE OR REPLACE TRIGGER Actualiza_SuministrosPieza  
  BEFORE UPDATE OF Codigo ON Pieza FOR EACH ROW  
BEGIN  
  UPDATE Suministros SET Pieza = :new.Codigo  
    WHERE Pieza = :old.Codigo;  
END Actualiza_SuministrosPieza;
```

Disparadores a Nivel de Fila: **WHEN**

- ▶ **Formato:** ... **FOR EACH ROW** **WHEN** <Condición_Disparo>
 - Sólo es válida en Disparadores a Nivel de Fila y siempre es opcional.
 - Si existe, el cuerpo del disparador se ejecutará sólo para las filas que cumplan la Condición de Disparo especificada.
 - En la Condición de Disparo pueden usarse los Pseudo-Registros `:old` y `:new`, pero en ese caso no se escribirán los dos puntos (:), los cuales son obligatorios en el cuerpo del disparador.
 - **Ejemplo:** Deseamos que los precios grandes no tengan más de 1 decimal. Si tiene 2 ó más decimales, redondearemos ese precio:

```
-- Si el Precio>200, redondearlos a un decimal:  
CREATE OR REPLACE TRIGGER Redondea_Precios_Grandes  
  BEFORE INSERT OR UPDATE OF Precio ON Pieza  
  FOR EACH ROW WHEN (new.Precio > 200)  
BEGIN  
  :new.Precio := ROUND (:new.Precio,1);  
END Redondea_Precios_Grandes;
```

- Se puede escribir ese disparador sin la cláusula **WHEN**, usando un **IF**:

```
... BEGIN  
    IF :new.Precio > 200 THEN  
        :new.Precio := ROUND (:new.Precio,1);  
    ... END IF;
```

Predicados INSERTING, DELETING, y UPDATING

- ▶ En los disparadores que se ejecutan cuando ocurren **diversas Operaciones DML (INSERT, DELETE o UPDATE)**, pueden usarse 3 **Predicados Booleanos** para conocer la operación disparadora:
 - **INSERTING** Vale **TRUE** si la orden de disparo es **INSERT**.
 - **DELETING** Vale **TRUE** si la orden de disparo es **DELETE**.
 - **UPDATING** Vale **TRUE** si la orden de disparo es **UPDATE**.
- ▶ **Ejemplo:** Guardar en una tabla de control la fecha, el usuario que modificó la tabla Empleados y el Tipo de Operación con la que modificó la tabla. Usando los Pseudo-Registros **:old** y **:new** también se pueden registrar los valores antiguos y los nuevos (si procede):

```
CREATE OR REPLACE TRIGGER Control_Empleados
AFTER INSERT OR DELETE OR UPDATE ON Empleados
BEGIN
    IF INSERTING THEN
        INSERT INTO Ctrl_Empleados (Tabla,Usuario,Fecha,Oper)
            VALUES ('Empleados', USER, SYSDATE, 'INSERT');
    ELSIF DELETING THEN
        INSERT INTO Ctrl_Empleados (Tabla,Usuario,Fecha,Oper)
            VALUES ('Empleados', USER, SYSDATE, 'DELETE');
    ELSE
        INSERT INTO Ctrl_Empleados (Tabla,Usuario,Fecha,Oper)
            VALUES ('Empleados', USER, SYSDATE, 'UPDATE');
    END IF;
END Control_Empleados;
```

Disparadores de Sustitución: INSTEAD OF

- ▶ Disparadores de Sustitución: Disparador que se ejecuta en lugar de la orden DML (ni antes ni después, sino sustituyéndola).
 - Cuando se intenta **modificar una vista** esta modificación puede no ser posible debido al formato de esa vista.
 - **Características**:
 - Sólo pueden definirse sobre Vistas.
 - Se activan en lugar de la Orden DML que provoca el disparo, o sea, la orden disparadora no se ejecutará nunca.
 - Deben tener Nivel de Filas.
 - Se declaran usando INSTEAD OF en vez de **BEFORE/AFTER**.

Disparadores de Sustitución: INSTEAD OF

- ▶ Ejemplo: Supongamos la siguiente vista:

```
CREATE VIEW Totales_por_Suministrador AS  
  SELECT Suministrador, MAX(Precio) Mayor, MIN(Precio) Menor  
  FROM Suministros SP, Pieza P  
  WHERE SP.Pieza = P.Codigo GROUP BY Suministrador;
```

- Disparador que borre un suministrador si se borra una tupla sobre esa vista:

```
CREATE OR REPLACE TRIGGER Borrar_en_Totales_por_Suministro  
  INSTEAD OF DELETE ON Totales_por_Suministrador  
  FOR EACH ROW  
BEGIN  
  DELETE FROM Suministradores  
    WHERE Nombre = :old.Suministrador;  
END Borrar_en_Totales_por_Suministro;
```

Disparadores: Observaciones

► Restricciones en las sentencias del cuerpo:

- **No puede emitir órdenes de Control de Transacciones: COMMIT, ROLLBACK o SAVEPOINT.**
 - El disparador se activa como parte de la orden que provocó el disparo y, por tanto, forma parte de la misma transacción. Cuando esa transacción es confirmada o cancelada, se confirma o cancela también el trabajo realizado por el disparador.
- **Cualquier Subprograma llamado por el disparador tampoco puede emitir órdenes de control de transacciones.**
- **Un disparador puede tener Restringido el acceso a ciertas tablas** → Dependiendo del tipo de disparador y de las restricciones que afecten a las tablas, dichas tablas pueden ser *mutantes (estar siendo modificadas por otra sesión, por ejemplo, Más después)*.

Disparadores: Observaciones

- ▶ **Diccionario de Datos:** Todos los datos de un **TRIGGER** están almacenados en la vista **USER_TRIGGERS**, en columnas como **OWNER**, **TRIGGER_NAME**, **TRIGGER_TYPE**, **TABLE_NAME**, **TRIGGER_BODY**...
- ▶ **Borrar un Disparador:**
 - **DROP TRIGGER** <NombreT> ;
- ▶ **Habilitar/Deshabilitar un Disparador,** sin necesidad de borrarlo:
 - **ALTER TRIGGER** <NombreT> {**ENABLE** | **DISABLE**} ;
 - Esto no puede hacerse con los subprogramas.

Problemas con actualización múltiple

- ▶ `CREATE TABLE p (p1 NUMBER CONSTRAINT pk_p_p1 PRIMARY KEY);`
- ▶ `CREATE TABLE f (f1 NUMBER CONSTRAINT fk_f_f1 REFERENCES p);`
- ▶ `CREATE TRIGGER pt AFTER UPDATE ON p
FOR EACH ROW
BEGIN
 UPDATE f SET f1 = :NEW.p1 WHERE f1 = :OLD.p1;
END pt;`
- ▶ `insert into p values (1);`
`insert into p values (2);`
`insert into p values (3);`
- ▶ `insert into f values (1);`
`insert into f values (2);`
`insert into f values (3);`
- ▶ `UPDATE p SET p1 = p1+1;`
- ▶ Al modificar el valor p1 en p también lo hace en f, de modo que en f los valores son 2, 2 y 3
- ▶ Cuando modifica el valor 2 en p, en f modifica todos aquellos que valgan 2, por lo que los valores son 3, 3 y 3
- ▶ Cuando modifica el 3 en p, en f todo termina valiendo 4

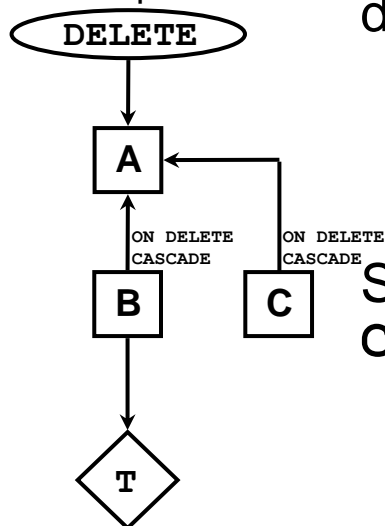
Disparadores: Tablas Mutantes

- ▶ **Tabla de Restricción o Tabla Padre** (*constraining table, Parent*): Tabla a la que referencia una llave externa en una **Tabla Hijo** (*Child*), por una Restricción de Integridad Referencial.
- ▶ **Tabla Mutante** (*mutating table*): Una tabla es mutante:
 - 1. Si está modificándose “actualmente” por una orden DML (INSERT, DELETE O UPDATE).
 - 2. Si está siendo leída por Oracle para forzar el cumplimiento de una restricción de integridad referencial.
 - 3. Si está siendo actualizada por Oracle para cumplir una restricción con ON DELETE CASCADE.
 - Ejemplo: Si la cláusula ON DELETE CASCADE aparece en la tabla Suministros, borrar una Pieza implica borrar todos sus Suministros.
 - Pieza (y Suministrador) es una Tabla de Restricción de Suministros.
 - Al borrar una pieza ambas tablas serán mutantes, si es necesario borrar suministros. Si se borra una pieza sin suministros sólo será mutante la tabla Pieza.
- ▶ **Un Disparador de Fila se define sobre una Tabla Mutante** (casi siempre).
 - En cambio, un **Disparador de Orden** NO (casi nunca): El disparador de Orden se ejecuta antes o después de la orden, pero no a la vez.

Disparadores: Tablas Mutantes

- ▶ Las Órdenes SQL en el cuerpo de un disparador tienen las siguientes 2 Restricciones por las que NO PUEDEN:
 - 1. Leer o Modificar ninguna Tabla Mutante de la orden que provoca el disparo.

Orden disparadora



Disparador de Fila T:
No puede acceder a las tablas A, B y C, con esa orden disparadora.

Esto incluye lógicamente la tabla de dicha orden y la tabla del disparador, que pueden ser distintas.

Ej.: Borrarnos una tupla de una tabla A. Esto implica borrar tuplas de una tabla B (que tiene una restricción **ON DELETE CASCADE** sobre A). Si este segundo borrado sobre B dispara algún disparador T, entonces, ese disparador no podrá ni leer ni modificar las tablas A y B pues son **mutantes**. Si lo hace se producirá un error.

Sus tablas de restricción afectadas por la cláusula **ON DELETE CASCADE**, también son mutantes.

Ej.: Si borrar en A implica borrar en una tercera tabla C, el disparador sobre B no podrá tampoco acceder a C, si ese disparador se activa por borrar sobre A (aunque en C no se borre nada). Sí podrá acceder a C si se activa por borrar directamente sobre B. También podría acceder a C si ésta no tuviera el **ON DELETE CASCADE** pero hay que tener en cuenta que NO se podrán borrar valores en A, si están siendo referenciados en C (ORA-2292).

- 2. Modificar las columnas de clave primaria, única o externa de una Tabla de Restricción a la que la tabla del disparador hacen referencia: Sí pueden modificarse las otras columnas.

¿Qué Tablas son Mutantes?

- ▶ Al escribir disparadores es importante responder a Dos Preguntas:

- ¿Qué Tablas son Mutantes?
- ¿Cuando esas Tablas Son mutantes?

Para responder a esa pregunta es necesario conocer el tipo de orden **DML** que se está ejecutando.

- ▶ ¿Qué Tablas son Mutantes?

- 1. Son mutantes las tablas afectadas por una operación **INSERT, DELETE o UPDATE**.
- 2. Si una **tabla Hijo** (p.e. Empleados) tiene un atributo clave externa (**Dpto**) a otra **tabla Padre** (Departamentos), **ambas** tablas serán mutantes si:
 - Insertamos (**INSERT**) en la tabla **Hijo**: Comprobar valores en la tabla padre.
 - Borramos (**DELETE**) de la tabla **Padre**: Impedir que tuplas hijas se queden sin padre.
 - Actualizamos (**UPDATE**) la tabla **Padre** o la tabla **Hijo**: Las 2 operaciones anteriores.
- 3. Si existe la restricción **ON DELETE CASCADE**, esta implica que si se borra de la **tabla Padre**, se borrarán las tuplas relacionadas en la **tabla Hijo** y, a su vez, pueden borrarse tuplas de tablas hijos de esa **tabla Hijo**, y así sucesivamente. En ese caso todas esas tablas serán mutantes.
 - En disparadores activados por un **DELETE**, es importante tener en cuenta si pueden ser activados por un borrado en cascada y, en ese caso no es posible acceder a todas esas tablas mutantes.

¿Cuándo son las Tablas Mutantes?

- ▶ Las 2 Restricciones anteriores en las órdenes SQL de un Disparador se aplican a:
 - Todos los Disparadores con Nivel de Fila.
 - Excepción: Cuando la orden disparadora es un **BEFORE INSERT** que afecta a una única fila, entonces esa tabla disparadora no es considerada como mutante.
 - Con las órdenes del tipo **INSERT INTO Tabla SELECT...** la tabla del disparador será mutante en ambos tipos de disparadores de Fila si se insertan varias filas.
 - Este es el único caso en el que un disparador de Fila puede leer o modificar la tabla del disparador.
 - Los Disparadores a Nivel de Orden cuando la orden de disparo se activa como resultado de una operación **ON DELETE CASCADE** (al borrar tuplas en la tabla Padre).
- ▶ Los **ERRORES** por Tablas Mutantes se detectan y se generan en Tiempo de Ejecución y no de Compilación (ORA-4091).

Tablas Mutantes: Disparador de Ejemplo

- Disparador que modifique el número de empleados de un departamento (columna `Departamentos.Num_Emp`) cada vez que sea necesario.
 - Ese número cambia al **INSERTAR** o **BORRAR** uno o más empleados, y al **MODIFICAR** la columna `Dpto` de la tabla `Empleados`, para uno o varios empleados.
 - La tabla `Departamentos` es una tabla de restricción de la tabla `Empleados`, pero el Disparador es correcto, porque modifica `Num_Emp`, que no es la llave primaria.
 - Este disparador no puede consultar la tabla `Empleados`, ya que esa tabla es mutante:
SELECT COUNT(*) INTO T FROM Empleados WHERE Dpto = :new.Dpto;

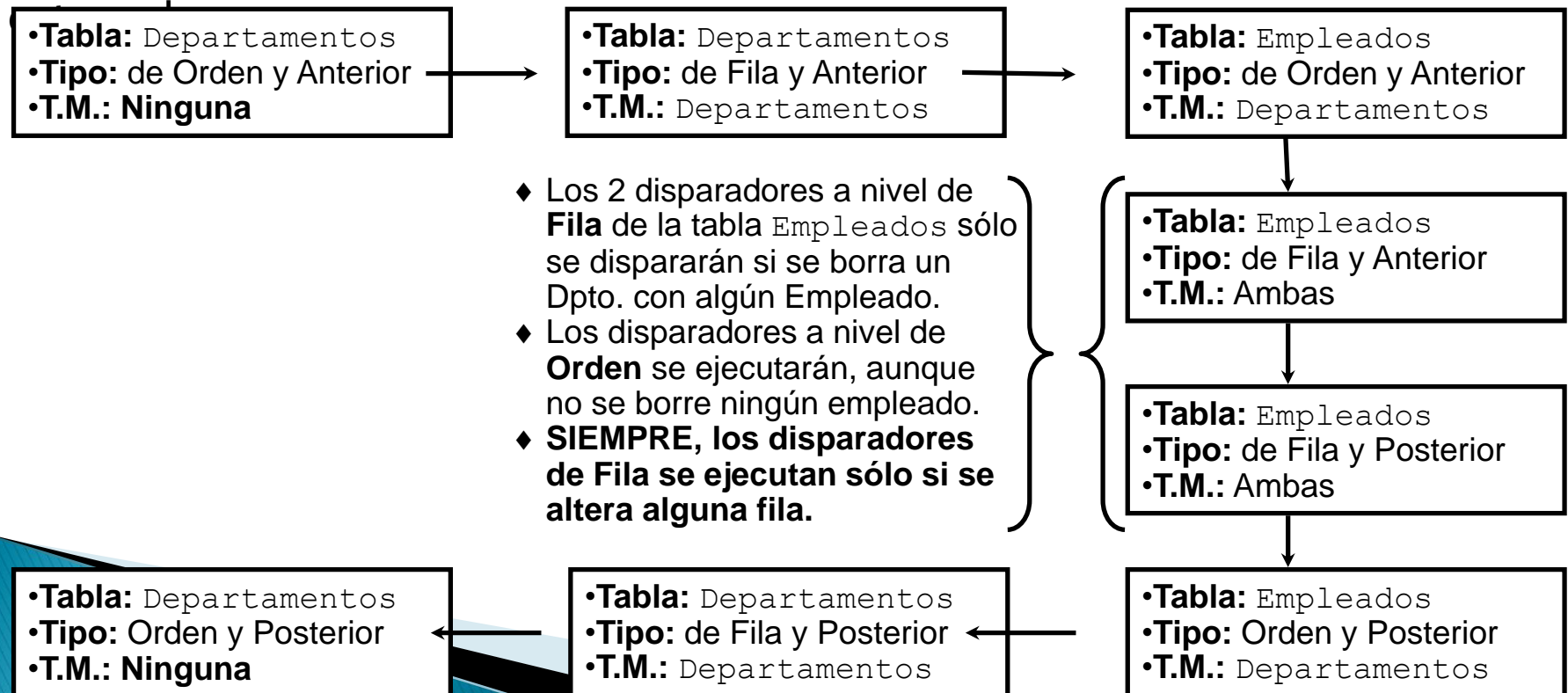
```
CREATE OR REPLACE TRIGGER Cuenta_Empleados
  BEFORE DELETE OR INSERT OR UPDATE OF Dpto ON Empleados
  FOR EACH ROW
BEGIN
  IF INSERTING THEN
    UPDATE Departamentos SET Num_Emp = Num_Emp+1
      WHERE NumDpto=:new.Dpto;
  ELSIF UPDATING THEN
    UPDATE Departamentos SET Num_Emp = Num_Emp+1
      WHERE NumDpto=:new.Dpto;
    UPDATE Departamentos SET Num_Emp = Num_Emp-1
      WHERE NumDpto=:old.Dpto;
  ELSE
    UPDATE Departamentos SET Num_Emp = Num_Emp-1
      WHERE NumDpto=:old.Dpto;
  END IF;
END;
```

Si en la tabla `Empleados` hubiera un **ON DELETE CASCADE** hacia `Departamentos`, entonces, al intentar borrar un Departamento habría que borrar sus empleados: Las dos tablas serían mutantes.

Al borrar sus empleados se dispararía este *trigger* que intentaría modificar la tabla `Departamentos` que sería mutante y daría ERROR.

Tablas Mutantes: Ejemplo Esquemático

- ▶ Sea una **tabla Padre** Departamentos, y una **Hijo** Empleados cuyo atributo llave externa es Dpto con la restricción **ON DELETE CASCADE** que fuerza a borrar todos los empleados de un departamento si su departamento es borrado.
- ▶ Supongamos que para la orden **DELETE** están implementados los 4 tipos de disparadores posibles (de fila y de orden, anterior y posterior) para las dos tablas.
- ▶ Si se ejecuta una orden **DELETE** sobre la **tabla Padre** Departamentos, se ejecutarán los siguientes disparadores, con las tablas mutantes indicadas, en



Tablas Mutantes: Esquema Padre/Hijo

Órdenes sobre Departamentos (Tabla Padre):

DELETE:

Los 4 disparadores de la tabla Hijo Empleados sólo se dispararán si se borra un Departamento que tenga algún Empleado.

INSERT: Su tabla Hijo no se ve afectada: No se disparan sus disparadores.

UPDATE: Su tabla Hijo no se ve afectada, porque sólo se permiten actualizar valores no referenciados en sus tablas Hijos (ORA-2292).

Tablas Mutantes: Esquema Padre/Hijo

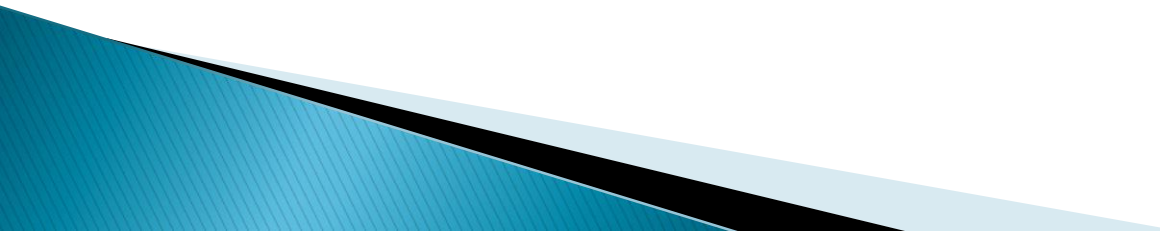
Órdenes sobre Empleados (Tabla Hijo): No se dispara ningún disparador del Padre.

DELETE: No afecta a la tabla Padre y sólo es mutante la Hijo.

INSERT:

◆ **Insertar una fila:** La tabla Hijo **no** es mutante en el disparador BEFORE INSERT (a pesar de ser la tabla del disparador) y **sí** lo es en el de **AFTER INSERT**.

◆ **Insertar varias filas:** La tabla Hijo es mutante en los dos disparadores de **Fila**.



Tablas Mutantes: Solución

► Solución al Problema de la Tabla Mutante:

- Las tablas mutantes surgen básicamente en los disparadores a nivel de Fila. Como en estos no puede accederse a las tablas mutantes, la **Solución es Crear Dos Disparadores:**
 - A Nivel de Fila: En este guardamos los valores importantes en la operación, pero no accedemos a tablas mutantes.
 - Estos valores pueden guardarse en:
 - Tablas de la BD especialmente creadas para esta operación.
 - Variables o tablas PL/SQL de un paquete: Como cada sesión obtiene su propia instancia de estas variables no tendremos que preocuparnos de si hay actualizaciones simultáneas en distintas sesiones.
 - A Nivel de Orden Posterior (AFTER): Utiliza los valores guardados en el disparador a Nivel de Fila para acceder a las tablas que ya no son mutantes.
 - Se usará un bucle que examine uno a uno todos los valores guardados en el trigger a nivel de fila:
 - Si se ha usado una tabla se deberá usar un cursor.

Otra Solución (Mejor): Compound Trigger

```
CREATE OR REPLACE TRIGGER  
  compound_trigger  
FOR UPDATE OF salario ON  
  empleados  
  COMPOUND TRIGGER
```

```
-- Parte Declarativa (opcional)  
-- Variables duran toda la  
sentencia que las dispara.
```

```
BEFORE STATEMENT IS  
  BEGIN  
    NULL;  
  END BEFORE STATEMENT;
```

```
BEFORE EACH ROW IS  
  BEGIN  
    NULL;  
  END BEFORE EACH ROW;
```

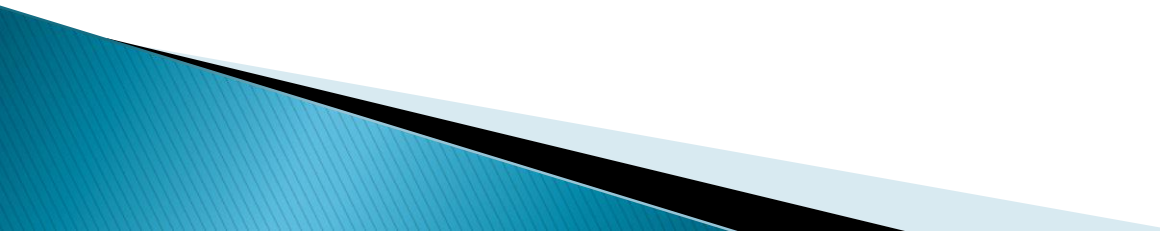
```
AFTER EACH ROW IS  
  BEGIN  
    NULL;  
  END AFTER EACH ROW;
```

```
AFTER STATEMENT IS  
  BEGIN  
    NULL;  
  END AFTER STATEMENT;  
END compound_trigger;
```

Ejemplo de trigger problemático

```
create or replace TRIGGER Print_current_max_salary  
  AFTER DELETE OR INSERT OR UPDATE ON  
employees  
  FOR EACH ROW
```

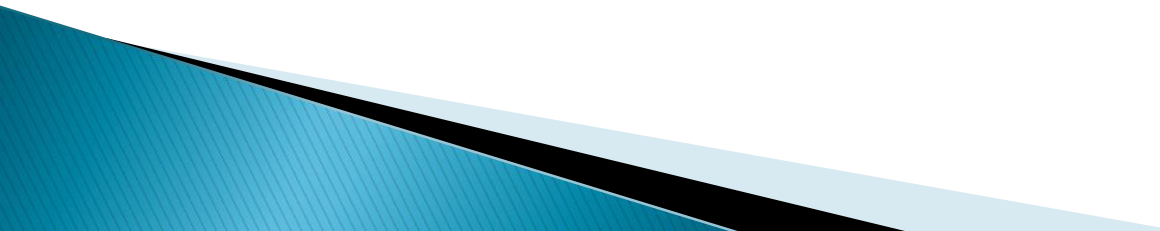
```
    sal_diff number;  
BEGIN  
    select max(salary) into sal_diff from employees;  
    dbms_output.put_line(' Max sueldo ' || sal_diff);  
-- Solo funciona en SQLPlus tras activar la salida con  
set serveroutput on  
END;
```



Con Compound Triggers?

```
create or replace TRIGGER Print_salary_changes  
for DELETE OR INSERT OR UPDATE ON employees  
COMPOUND TRIGGER
```

```
    sal_diff number;  
before statement is  
BEGIN  
    select max(salary) into sal_diff from employees;  
end before statement;  
before each row is  
begin  
    dbms_output.put_line(' Max sueldo ' || sal_diff);  
end before each row;  
END;
```



Triggers sobre sentencias DDL y eventos

▶ DDL:

- BEFORE CREATE y AFTER CREATE
- BEFORE ALTER y AFTER ALTER
- BEFORE DROP y AFTER DROP

▶ Eventos de la Base de Datos:

- SERVERERROR, LOGON, LOGOFF, STARTUP, SHUTDOWN

```
CREATE OR REPLACE TRIGGER trigger_ejemplo1
```

```
AFTER LOGON ON DATABASE
```

```
BEGIN
```

```
    INSERT INTO tabla_ejemplo1 VALUES(user, sysdate);
```

```
END;
```

```
CREATE OR REPLACE TRIGGER trigger_ejemplo2
```

```
ON DATABASE SHUTDOWN
```

```
BEGIN
```

```
    INSERT INTO tabla_ejemplo2 VALUES(sysdate);
```

```
END;
```

Trabajos: Jobs

- ▶ Oracle Scheduler: planificador de trabajos implementado en el paquete PL/SQL DBMS_SCHEDULER
- ▶ JOB = PROGRAM + SCHEDULE
- ▶ ¿Qué se puede ejecutar?:
 - Unidades de **programa** de bases de datos (STORED_PROCEDURE, PLSQL_BLOCK)
 - **Programas** externos (ejecutables, scripts, etc.) de forma local o en otras bases de datos (EXTERNAL)
- ▶ ¿Cuándo se ejecuta?:
 - Time-based scheduling. Se define la fecha y la repetición
 - Event-based scheduling. Cuando falla una transacción, llega un fichero, etc.
 - Dependency scheduling → Chains. Se pueden definir cadenas complejas de trabajos, con ramas, etc.

Ejecución de los trabajos

Una Trabajo es un objeto y el dueño es el usuario que lo crea

El programa se ejecuta con las credenciales del trabajo (salvo que se indique lo contrario)

- ▶ Para crear un trabajo hay que tener el role CREATE JOB

Al ejecutarse:

- ▶ Cuando es hora de ejecutarse, se despierta un proceso esclavo para ejecutar el programa
 1. Se recogen todos los metadatos necesarios, argumentos de programa e información de los privilegios
 2. Se crea una sesión con las credenciales del dueño del trabajo, se comienza una transacción y se ejecuta el programa del trabajo.
 3. Cuando se completa, se hace commit y se termina la transacción.
 4. Se cierra la sesión.

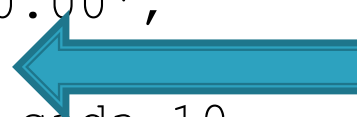
Cuando el trabajo termina:

- ▶ Se planifica la siguiente ejecución si es necesario.
- ▶ Se modifica la tabla de trabajos para indicar si ha terminado o si se tiene que ejecutar de nuevo.
- ▶ Se inserta una entrada en la tabla de log de trabajos.
 - USER_SCHEDULER_JOB_LOG

Creación de Trabajos

► Un ejemplo sencillo:

```
BEGIN
  DBMS_SCHEDULER.CREATE_JOB (
    job_name      => 'mi_tarea',
    job_type      => 'PLSQL_BLOCK',
    job_action    => 'BEGIN insert into prueba values
(17,USER,sysdate); END;',
    start_date    => SYSDATE+1,
    repeat_interval => 'FREQ=SECONDLY;INTERVAL=10',
    end_date      => '30/MAY/2018 20.00.00',
    enabled       => TRUE,
    comments      => 'Inserta una tupla cada 10
segundos a partir de mañana y hasta el 30 de mayo de
2018');
END;
```



- Cuidado, por defecto, deshabilitado

Gestión de Trabajos

- ▶ Ver los trabajos:
 - `select * from user_scheduler_jobs;`
- ▶ Borrar un trabajo (o varios):
 - `DBMS_SCHEDULER.DROP_JOB ('job1, job2, job3');`
- ▶ Parar un trabajo (no su planificación):
 - `DBMS_SCHEDULER.STOP_JOB ('job1, job2, job3');`
- ▶ Deshabilitarlo:
 - `DBMS_SCHEDULER.DISABLE ('job1, job2, job3');`
- ▶ Habilitarlo:
 - `DBMS_SCHEDULER.ENABLE ('job1, job2, job3');`

Ejecución de un procedimiento con argumentos

```
DBMS_SCHEDULER.CREATE_JOB (  
    job_name           =>  'otra_tarea',  
    job_type           =>  'STORED_PROCEDURE',  
    job_action         =>  'MI_PROCEDIMIENTO',  
    number_of_arguments =>  2,  
    start_date         =>  sysdate,  
    repeat_interval    =>  'FREQ=DAILY;BYHOUR=22,23',  
    end_date           =>  null,  
    auto_drop          =>FALSE,  
    enabled            =>FALSE,  
    comments           =>  'Se ejecuta 2 veces al dia, a las 10 y 11 pm');
```

```
DBMS_SCHEDULER.SET_JOB_ARGUMENT_VALUE('otra_tarea',1,'Valor1');  
DBMS_SCHEDULER.SET_JOB_ARGUMENT_VALUE('otra_tarea',2,'Valor2');  
DBMS_SCHEDULER.ENABLE('otra_tarea');
```

Ejemplos de Planificación

- ▶ Cada viernes:
 - `FREQ=WEEKLY; BYDAY=FRI;`
- ▶ Cada 2 viernes:
 - `FREQ=WEEKLY; INTERVAL=2; BYDAY=FRI;`
- ▶ El último día de cada mes
 - `FREQ=MONTHLY; BYMONTHDAY=-1;`
- ▶ El penúltimo día de cada mes:
 - `FREQ=MONTHLY; BYMONTHDAY=-2;`
- ▶ Cada año, el 10 de marzo:
 - `FREQ=YEARLY; BYMONTH=MAR; BYMONTHDAY=10;`