Otros Objetos de la Base de Datos

Disparadores y Trabajos

Definición de TRIGGER

- Son elementos que especifican acciones desencadenadas por una operación sobre:
  - Operación DML: INSERT, DELETE o UPDATE
  - Operación DDL o evento de la Base de datos
- Al contrario que un procedimiento o función no son llamados sino que se ejecutan de forma implícita.
- Un Disparador No admite Argumentos
- Vista USER_TRIGGERS

Utilidad de los TRIGGERS

- 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.

Creación

```
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.**
  - » Si la orden **UPDATE** lleva una lista de atributos el Disparador sólo se ejecutará si se actualiza alguno de ellos**: UPDATE OF <Lista_Atributos>**

Creación

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

- 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 Tabla o a Nivel de Fila (**FOR EACH ROW**).
  - **Nivel de Tabla**: Se activan sólo una vez, antes o después de la **Sentencia** u operación **DML**.
  - **Nivel de Fila**: Se activan una vez por cada **Fila** afectada por la operación **DML** (una misma operación **DML** puede afectar a varias filas).

Creación

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

- **<Bloque_del_TRIGGER>**
  - El cuerpo es un bloque de PL/SQL.
  - No tiene órdenes de control de transacciones. Idem para rutinas llamadas por el disparador.
  - No se pueden declarar variables de tipo LONG o LONG RAW.
  - No hay libertad absoluta de acceso a tablas (problema de la tabla mutante).
  - Pueden usarse los predicados INSERTING, UPDATING o DELETING en el cuerpo para discriminar el suceso.

Ejemplo

- **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 tabla)

```
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;
```

Orden de Ejecución

1. Ejecutar el disparador tipo BEFORE a nivel de tabla.
2. Para cada fila a la que afecte la orden
   a) Ejecutar 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 disparador AFTER a nivel de fila, sólo si dicha fila cumple la condición de la cláusula WHEN (si existe).
3. Ejecutar el disparador tipo AFTER a nivel de orden.

Variables de acoplamiento

Cuando el Trigger es de nivel de fila se pueden utilizar 2 variables de acoplamiento, `:old` y `:new`

	:old			:new		
DELETE	IB	456	VLC	IB	456	VLC
	IB	875	AGP			
	AA	456	AGP	AA	456	AGP
UPDATE	IB	456	VLC	IB	456	VLC
	IB	875	AGP	IB	999	VLC
	AA	456	AGP	AA	456	AGP
INSERT	IB	456	VLC	IB	456	VLC
				IB	875	AGP
	AA	456	AGP	AA	456	AGP

Ejemplos

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;
```

Ejemplos

- Guardar en una tabla de control la fecha y el usuario que modifica la tabla Asignaturas. Se guarda una sola fila por cada sentencia INSERT, independientemente del número de filas involucradas.
- Creamos la tabla

```
CREATE TABLE Ctr_Tablas
   Usuario VARCHAR2(30),
   Fecha DATE,
   Tabla VARCHAR2(30),
   Operacion VARCHAR2(30) );
```

Ejemplos

```
CREATE OR REPLACE TRIGGER Control_Asignaturas
AFTER INSERT OR DELETE OR UPDATE ON Asignaturas
BEGIN
   IF INSERTING THEN -- Se ejecuta solo si es de INSERT
       INSERT INTO Ctrl_Tablas (Tabla,Usuario,Fecha,Operacion)
       VALUES ('Asignaturas', USER, SYSDATE, 'INSERT');
   ELSIF DELETING THEN
       INSERT INTO Ctrl_Tablas (Tabla,Usuario,Fecha, Operacion)
       VALUES ('Asignaturas', USER, SYSDATE, 'DELETE');
   ELSE    -- UPDATING
       INSERT INTO Ctrl_Tablas (Tabla,Usuario,Fecha, Operacion)
       VALUES ('Asignaturas', USER, SYSDATE, 'UPDATE');
   END IF;
END Control Asignaturas;
```

Disparadores de Sustitución

- Disparador que se ejecuta en lugar de la orden DML (ni antes ni después, sino sustituyéndola).
  - **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`**.

Ejemplos

- Queremos crear una vista que guarde información agrupada del número de alumnos matriculados en cada asignatura
- Creamos la vista

```
CREATE VIEW Global_Matriculas AS
SELECT asignatura, COUNT(*) Num_Matriculas
FROM Matricular
GROUP BY asignatura;
```

Ejemplos

- Se quiere crear un disparador que borre todas las matrículas de una asignatura si se borra una tupla sobre la vista Global Matriculas

```
CREATE OR REPLACE TRIGGER Borrar_en_Global
INSTEAD OF DELETE ON Global_Matriculas
FOR EACH ROW
BEGIN
   DELETE FROM matriculas
   WHERE asignatura = :OLD.asignatura;
END Borrar_en_Global;
```

Jobs

- Oracle Scheduler: planificador de trabajos `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)

Al ejecutarse:

- Cuando es hora de ejecutarse, se despierta un proceso 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

Ejemplo

```
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;
```

Gestión de trabajos

- Ver los trabajos:
  - select * from user_scheduler_jobs;
- Borrar un trabajo (o varios):
  - DBMS_SCHEDULER.DROP_JOB ('job1, job2, job3');
- Deshabilitarlo:
  - DBMS_SCHEDULER.DISABLE ('job1, job2, job3');
- Habilitarlo:
  - DBMS_SCHEDULER.ENABLE ('job1, job2, job3');

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;