Facultad: Ingeniería
Escuela: Computación
Asignatura: Base de datos II

Tema: DISPARADORES (TRIGGERS)

Objetivo

Conocer la definición y utilización de los triggers

Materiales

- Oracle 12 c
- Guía Número 7

Introducción

DISPARADORES

Un disparador define una acción que la base de datos debe llevar a cabo cuando se produce algún suceso relacionado con la misma. Los disparadores (triggers) pueden utilizarse para completar la integridad referencial, también para imponer reglas de negocio complejas o para auditar cambios en los datos. El código contenido en un disparador, denominado cuerpo del disparador, está formado por bloques PL/SQL. La ejecución de disparadores es transparente al usuario.

Existen varios tipos de disparadores, dependiendo del tipo de transacción de disparo y el nivel en el que se ejecuta el disparador (trigger):

1.- Disparadores de nivel de fila: se ejecutan una vez para cada fila afectada por una instrucción DML. Los disparadores de nivel de fila se crean utilizando la cláusula for each row en el comando create trigger.

2.- Disparadores de nivel de instrucción: se ejecutan una vez para cada intrucción DML. Por ejemplo, si una única intrucción INSERT inserta 500 filas en una tabla un disparador de nivel de instrucción para dicha tabla sólo se ejecutará una vez. Los disparadores de nivel de instrucción son el tipo predeterminado que se crea con el comando create trigger.

3.- Disparadores Before y After: puesto que los disparadores son ejecutados por sucesos, puede establecerse que se produzcan inmediatamente antes (before) o después (after) de dichos sucesos.

4.- Disparadores Instead Of: puede utilizar INSTEAD OF para indicar a Oracle lo que tiene que hacer en lugar de realizar las acciones que invoca el disparador. Por ejemplo, podría usar un disparador

INSTEAD OF en una vista para gestionar las inserciones en una tabla o para actualizar múltiples tablas que son parte de una vista.

5.- Disparadores de esquema: puede crear disparadores sobre operaciones en el nivel de esquema tales como create table, alter table, drop table, audit, rename, truncate y revoke. Puede incluso crear disparadores para impedir que los usuarios eliminen sus propias tablas. En su mayor parte, los disparadores de nivel de esquema proporcionan dos capacidades: impedir operaciones DDL y proporcionar una seguridad adicional que controle las operaciones DDL cuando éstar se producen.

6.- Disparadores en nivel de base de datos: puede crear disparadores que se activen al producirse sucesos de la base de datos, incluyendo errores, inicios de sesión, conexiones y desconexiones. Puede utilizar este tipo de disparador para automatizar el mantenimiento de la base de datos o las acciones de auditoría.

Sintaxis

```
DELIMITER |
  CREATE  TRIGGER nombre
    {BEFORE | AFTER | INSTEAD OF} // Temporalidad del Evento
    {INSERT | DELETE | UPDATE [OF <lista de columnas>]} ON <tabla>
    [FOR EACH ROW]  //Granularidad
    [WHEN condición]
    [DECLARE //Declaración de variables
      . . .]
  BEGIN
    cuerpo del trigger
    [EXCEPTION
      . . .]
  END
  |
  DELIMITER ;
```

Para eliminar

```
DROP TRIGGER nombre_disparador;
```

Activar/ Desactivar dispadores

Existen dos opciones.

- ALTER TRIGGER nombre_disparador {DISABLE | ENABLE};

- ALTER TABLE nombre_tabla {ENABLE | DISABLE} ALL TRIGGERS;

La ejecución del disparador puede ser antes (before) o después (after) de llevar a cabo la sentencia disparadora. Es posible especificar condiciones adicionales para la ejecución del disparador (restrictores).

Dado que una sentencia disparadora puede afectar una o más filas de una tabla, es necesario especificar si se quiere que el disparador se ejecute para cada una de las filas afectadas o para el bloque en general.

Para diseñar un disparador hay que cumplir dos requisitos:

- Especificar las condiciones en las que se va a ejecutar el disparador. Esto se descompone en un evento que causa la comprobación del disparador y una condición que se debe cumplir para ejecutar el disparador.
- Especificar las acciones que se van a realizar cuando se ejecute el disparador.

Registros old y new.

Estas variables se utilizan del mismo modo que cualquier otra variable, con la salvedad de que no es necesario declararlas,son de tipo %ROWTYPE y contienen una copia del registro antes (OLD) y después (NEW) de la acción SQL (INSERT, UPDATE, DELETE) que ha ejecutado el trigger. Utilizando esta variable podemos acceder a los datos que se están insertando, actualizando o borrando.

De uso exclusivo en los disparadores de nivel de fila, si se intenta hacer referencia a cualquiera de los dos dentro de otro tipo de disparador, se obtendrá un error de compilación.

La siguiente tabla resume los valores regresados por estos seudoregistros en diferentes eventos

Evento	**Seudoregistros**	
	:OLD	**:NEW**
INSERT	NULL	Nuevos valores
DELETE	Valores almacenados	NULL
UPDATE	Valores almacenados	Nuevos valores

PROCEDIMIENTO

Ejemplo 1, cree la siguiente estructura

```
Conectado a:
Oracle Database 12c Standard Edition Release 12.2.0.1.0 - 64bit Production

SQL>  CREATE TABLE amigos (
  2     idAmigo NUMBER(4) PRIMARY KEY,
  3     nombre  CHAR(35) NOT NULL,
  4     celular NUMBER(13) NOT NULL
  5   );

Tabla creada.

SQL>
SQL>   CREATE TABLE amigosCopia (
  2     idAmigo NUMBER(4) PRIMARY KEY,
  3     nombre  CHAR(35) NOT NULL,
  4     celular NUMBER(13) NOT NULL
  5   );

Tabla creada.

SQL>
SQL>  CREATE SEQUENCE idAmigo INCREMENT BY 1;
```

Ahora cree el siguiente disparador

```
CREATE OR REPLACE TRIGGER afterAmigos
 AFTER INSERT ON amigos
 REFERENCING NEW AS NEW
 FOR EACH ROW
 BEGIN
   INSERT INTO amigosCopia VALUES (:NEW.idAmigo,:NEW.nombre, :NEW.celular);
       DBMS_OUTPUT.PUT_LINE('Registro duplicado en amigosCopia.');
 END;
 /
```

Inserte los siguientes datos

```
INSERT INTO amigos VALUES (idAmigo.NEXTVAL, 'RAUL TRUJILLO JIMENEZ', 9345678);
INSERT INTO amigos VALUES (idAmigo.NEXTVAL, 'MARTIN ARIAS PEÑA', 9344678);
```

Revise la tabla amigoscopia, el disparador realizó un insert al momento de insertar datos en la tabla amigos

```
SQL> select * from amigoscopia;

   IDAMIGO NOMBRE                                CELULAR
---------- ----------------------------------- ----------
         1 RAUL TRUJILLO JIMENEZ                9345678
         2 MARTIN ARIAS PEÑA                    9344678
```

Manejo de excepciones

Para hacer que un trigger ejecute un ROLLBACK de la transacción que tiene activa y teniendo en cuenta que en las sentencias que componen el cuerpo de un trigger no puede haber este tipo de sentencias (rollback, commit,…) hay que ejecutar '*error / excepcion*' mediante la sentencia raise_application_error cuya sintaxis es:

```
RAISE_APPLICATION_ERROR(num_error,'mensaje');
```

Donde *num_error* en el rango: [-20000 y -20999]

Ejemplo: Retomando el ejemplo anterior, desarrollamos una excepción en nuestra tabla amigos. En la cual no admitimos como amiga a 'Isabel Mebarak Ripoll'

```
CREATE OR REPLACE TRIGGER noAmigos
BEFORE INSERT ON amigos
REFERENCING NEW AS NEW
  FOR EACH ROW
BEGIN
  IF :NEW.nombre = 'Isabel Mebarak Ripoll' THEN
    RAISE_APPLICATION_ERROR(-20000,'Esta persona no puede ser aceptada en la tabla amigos');
  END IF;
END;
/
```

Ejemplo 2, cree las siguientes tablas,

```
SQL> create table libros(
  2    codigo number(6),
  3    titulo varchar2(40),
  4    autor varchar2(30),
  5    editorial varchar2(20),
  6    precio number(6,2)
  7   );

Tabla creada.

SQL>
SQL>  create table control(
  2    usuario varchar2(30),
  3    fecha date,
  4    codigo number(6),
  5    precioanterior number(6,2),
  6    precionuevo number(6,2)
  7   );

Tabla creada.
```

Ingresamos los siguientes registros en la tabla libros

insert into libros values(100,'Uno','Richard Bach','Planeta',25);

insert into libros values(103,'El aleph','Borges','Emece',28);

insert into libros values(105,'Matematica estas ahi','Paenza','Nuevo siglo',12);

insert into libros values(120,'Aprenda PHP','Molina Mario','Nuevo siglo',55);

insert into libros values(145,'Alicia en el pais de las maravillas','Carroll','Planeta',35);

Creamos un trigger a nivel de fila que se dispara "antes" que se ejecute un "update" sobre el campo "precio" de la tabla "libros". En el cuerpo del disparador se debe ingresar en la tabla "control", el nombre del usuario que realizó la actualización, la fecha, el código del libro que ha sido modificado, el precio anterior y el nuevo:

```
create or replace trigger tr_actualizar_precio_libros
before update of precio
on libros
for each row
begin
 insert into control values(user,sysdate,:new.codigo,:old.precio,:new.precio);
end tr_actualizar_precio_libros;
```

Cuando el trigger se dispare, antes de ingresar los valores a la tabla, almacenará en "control", además del nombre del usuario y la fecha, el precio anterior del libro y el nuevo valor.

Actualizamos el precio del libro con código 100:

update libros set precio=30 where codigo=100;

Veamos lo que se almacenó en "control" al dispararse el trigger:

select *from control;

Los campos "precioanterior" y "precionuevo" de la tabla "control" almacenaron los valores de ":old.precio" y ":new.precio" respectivamente.

Actualizamos varios registros:

update libros set precio=precio+precio*0.1 where editorial='Planeta';

Veamos lo que se almacenó en "control" al dispararse el trigger:

select *from control;

Los campos "precioanterior" y "precionuevo" de la tabla "control" almacenaron los valores de ":old.precio" y ":new.precio" respectivamente de cada registro afectado por la actualización.

Modificamos la editorial de un libro:

update libros set editorial='Sudamericana' where editorial='Planeta';

El trigger no se disparó, pues fue definido para actualizaciones del campo "precio" unicamente.

Ejemplo 3:

Vamos a reemplazar el trigger anteriormente creado. Ahora el disparador "tr_actualizar_precio_libros" debe controlar el precio que se está actualizando, si supera los 50 dolares, se debe redondear tal valor a entero hacia abajo (empleando "floor"), es decir, se modifica el valor ingresado accediendo a ":new.precio" asignándole otro valor:

```
create or replace trigger tr_actualizar_precio_libros
 before update of precio
 on libros
 for each row
 begin
  if (:new.precio>50) then
   :new.precio:=floor(:new.precio);
  end if;
  insert into control values(user,sysdate,:new.codigo,:old.precio,:new.precio);
 end tr_actualizar_precio_libros;
/
```

Vaciamos la tabla "control":

truncate table control;

Actualizamos el precio del libro con código 100:

update libros set precio=54.99 where codigo=100;

Veamos cómo se actualizó

select *from libros where codigo=100;

El nuevo precio actualizado se redondeó a 54.

Veamos lo que se almacenó en "control" al dispararse el trigger:

select *from control;

Ejemplo 4

Creamos un disparador para múltiples eventos, que se dispare al ejecutar "insert", "update" y "delete" sobre "libros". En el cuerpo del trigger se realiza la siguiente acción: se almacena el nombre del usuario, la fecha y los antiguos y viejos valores de "precio":

```
create or replace trigger tr_libros
 before insert or update or delete
 on libros
 for each row
 begin
  insert into control values(user,sysdate,:old.codigo,:old.precio,:new.precio);
 end tr_libros;
```

Realice las operaciones de insert, delete y update y verifique el funcionamiento del trigger

Predicados condicionales

Cuando se crea un trigger para más de una operación DML, se puede utilizar un predicado condicional en las sentencias que componen el trigger que indique que tipo de operación o sentencia ha disparado el trigger. Estos predicados condicionales son los siguientes:

Inserting: Retorna true cuando el trigger ha sido disparado por un INSERT

Deleting: Retorna true cuando el trigger ha sido disparado por un DELETE

Updating: Retorna true cuando el trigger ha sido disparado por un UPDATE

Updating (columna): Retorna true cuando el trigger ha sido disparado por un UPDATE y la columna ha sido modificada

```
CREATE TABLE pokemon (
  idPokemon  INTEGER PRIMARY KEY,
  nombre     CHAR(30) NOT NULL,
  generacion INTEGER NOT NULL);
  CREATE SEQUENCE idPokemon INCREMENT BY 1;
```

```
INSERT INTO pokemon VALUES(idPokemon.NEXTVAL,'Articuno',1);
INSERT INTO pokemon VALUES(idPokemon.NEXTVAL,'Zapdos',1);
INSERT INTO pokemon VALUES(idPokemon.NEXTVAL,'Moltres',1);
INSERT INTO pokemon VALUES(idPokemon.NEXTVAL,'Mewtwo',1);
INSERT INTO pokemon VALUES(idPokemon.NEXTVAL,'Mew',1);
INSERT INTO pokemon VALUES(idPokemon.NEXTVAL,'Raikou',2);
INSERT INTO pokemon VALUES(idPokemon.NEXTVAL,'Entei',2);
INSERT INTO pokemon VALUES(idPokemon.NEXTVAL,'Suicune',2);
INSERT INTO pokemon VALUES(idPokemon.NEXTVAL,'Lugia',2);
INSERT INTO pokemon VALUES(idPokemon.NEXTVAL,'Ho-Oh ',2);
INSERT INTO pokemon VALUES(idPokemon.NEXTVAL,'Celebi',2);
INSERT INTO pokemon VALUES(idPokemon.NEXTVAL,'Regirock',3);
INSERT INTO pokemon VALUES(idPokemon.NEXTVAL,'Regice',3);
INSERT INTO pokemon VALUES(idPokemon.NEXTVAL,'Registeel',3);
INSERT INTO pokemon VALUES(idPokemon.NEXTVAL,'Latias',3);
INSERT INTO pokemon VALUES(idPokemon.NEXTVAL,'Uxie',4);
INSERT INTO pokemon VALUES(idPokemon.NEXTVAL,'Mesprit',4);
INSERT INTO pokemon VALUES(idPokemon.NEXTVAL,'Azelf',4);
```

Disparador

```
SQL> CREATE  OR REPLACE TRIGGER auditoria
  2    BEFORE INSERT OR DELETE OR UPDATE ON pokemon
  3    FOR EACH ROW
  4    DECLARE
  5        operacion CHAR(1);
  6    BEGIN
  7      /* Usemos 'I' para INSERT, 'D' para DELETE, y 'U' para UPDATE. */
  8      IF INSERTING THEN
  9         operacion := 'I';
 10      ELSIF UPDATING THEN
 11        operacion := 'U';
 12      ELSE
 13        operacion := 'D';
 14      END IF;
 15
 16      DBMS_OUTPUT.put_line(operacion ||' '|| USER ||' ' ||SYSDATE);
 17    END;
 18    /

Disparador creado.
```

Verifique su funcionamiento

```
SQL> SET SERVEROUT ON;
SQL> UPDATE pokemon SET generacion = 5 WHERE generacion = 4 ;
U SYSTEM 18/09/17
U SYSTEM 18/09/17
U SYSTEM 18/09/17

3 filas actualizadas.

SQL> DELETE pokemon WHERE generacion = 5;
D SYSTEM 18/09/17
D SYSTEM 18/09/17
D SYSTEM 18/09/17

3 filas suprimidas.
```

ANALISIS DE RESULTADOS

1- Cree un disparador que se active cuando modificamos algún campo de "empleados" y almacene en "controlCambios" el nombre del usuario que realiza la actualización, la fecha, el dato que se cambia y el nuevo valor:

```
create table empleados(
 documento char(8) not null,
 nombre varchar2(30) not null,
 domicilio varchar2(30),
 seccion varchar2(20)
);
```

```
create table controlCambios(
 usuario varchar2(30),
 fecha date,
 datoanterior varchar2(30),
 datonuevo varchar2(30)
);


insert into empleados values('22222222','Ana Acosta','Bulnes 56','Secretaria');
 insert into empleados values('23333333','Bernardo Bustos','Bulnes 188','Contaduria');
 insert into empleados values('24444444','Carlos Caseres','Caseros 364','Sistemas');
 insert into empleados values('25555555','Diana Duarte','Colon 1234','Sistemas');
 insert into empleados values('26666666','Diana Duarte','Colon 897','Sistemas');
 insert into empleados values('27777777','Matilda Morales','Colon 542','Gerencia');
```

2- Crear un desencadenador que se active cuando ingresamos un nuevo registro en "empleados", debe almacenar en "controlCambios" el nombre del usuario que realiza el ingreso, la fecha, "null" en "datoanterior" (porque se dispara con una inserción) y en "datonuevo" el documento:

3- crear trigger sobre "empleados" que se active cuando eliminamos un registro en "empleados", debe almacenar en "controlCambios" el nombre del usuario que realiza la eliminación, la fecha, el documento en "datoanterior" y "null" en "datonuevo"