**TRIGGERS**

Un trigger es una herramienta para mantener la integridad de las tablas en Oracle.

Permite incluir código PL-SQL para poder comprobar las características de las consultas de acción.

Como integridad de datos en Oracle tenemos una serie de herramientas ya conocidas, como son las restricciones: Primary Key, Foreign Key, Check.

Cuando dichas restricciones se quedan cortas y tienen que ver más con la lógica de nuestra aplicación, es cuando debemos utilizar Triggers.

Un Trigger se activa con las consultas de acción: INSERT, UPDATE, DELETE.

Es un código que nos permite “interceptar” los datos que se están manejando en las consultas de acción.

Los Triggers sirven para bloquear acciones, no sirven para **commit** o **rollback**, de hecho, no podemos hacer dichas acciones en un Trigger.

Tenemos varios tipos de Trigger:

* BEFORE, AFTER, INSTEAD OF

Before y After se utilizan sobre las tablas.

Instead Of se utiliza sobre las vistas.

Dentro de un trigger, podemos configurar para que se realice con condiciones (WHEN) o, por ejemplo, para que se active con cada fila de una consulta.

No podemos tener más de un Trigger al mismo tiempo (before o after) para la misma acción y misma fila.

Sintaxis:

CREATE OR REPLACE TRIGGER nombre\_trigger

BEFORE | AFTER

UPDATE | DELETE | INSERT

ON TABLA

[ FOR EACH ROW ]

DECLARE

BEGIN

--CODIGO PL-SQL

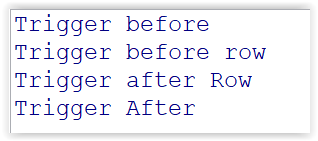
END;

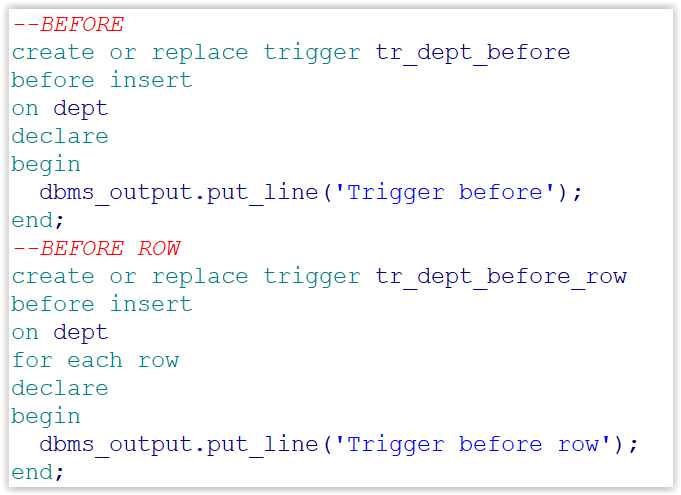
No podemos bloquear una acción en un Trigger. Ya sea un Trigger Before o After.

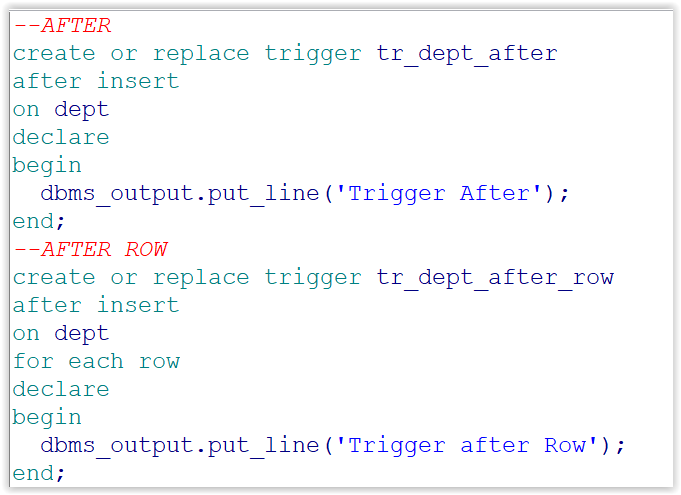
Si nosotros queremos bloquear una acción que ha ejecutado el trigger, se utiliza un **lanzador de errores**

RAISE\_APPLICATION\_ERROR (-20001, ' Error ');

Ejemplo de Orden de Triggers.







--BEFORE

create or replace trigger tr\_dept\_before

before insert

on dept

declare

begin

dbms\_output.put\_line('Trigger before');

end;

--BEFORE ROW

create or replace trigger tr\_dept\_before\_row

before insert

on dept

for each row

declare

begin

dbms\_output.put\_line('Trigger before row');

end;

--AFTER

create or replace trigger tr\_dept\_after

after insert

on dept

declare

begin

dbms\_output.put\_line('Trigger After');

end;

--AFTER ROW

create or replace trigger tr\_dept\_after\_row

after insert

on dept

for each row

declare

begin

dbms\_output.put\_line('Trigger after Row');

end;

**CAPTURAR DATOS DE LAS CONSULTAS DE ACCION CON TRIGGERS**

Un Trigger nos permite interceptar una consulta de acción realizada en Oracle.

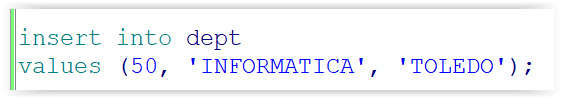
Tenemos unas variables ROWTYPE que nos permiten poder acceder a la información incluida en la consulta de acción.

Dichas variables son **:NEW** y **:OLD**

Dichas variables solamente podemos utilizarlas dentro de un Trigger de tipo **FOR EACH ROW**.

Cada una de las variables contiene la información dependiendo de la acción del Trigger.

* **INSERT**: Solamente tiene información :NEW

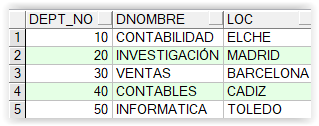


:new.dept\_no 🡪 50

:new.dnombre 🡪 INFORMATICA

:new.loc 🡪 TOLEDO

* DELETE: Solamente tiene información :OLD



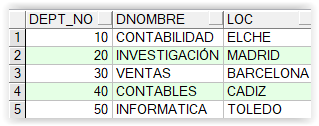


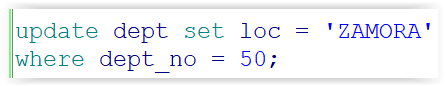
:old.dept\_no 🡪 50

:old.dnombre 🡪 INFORMATICA

:old.loc 🡪 TOLEDO

* UPDATE: Tenemos información en las dos variables:
  + :NEW estarán los cambios de los SET
  + :OLD están los datos que había en la tabla





:new.dept\_no 🡪 50

:new.dnombre 🡪 INFORMATICA

:new.loc 🡪 ZAMORA

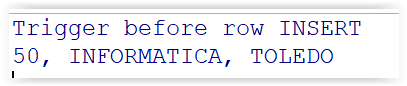
:old.dept\_no 🡪 50

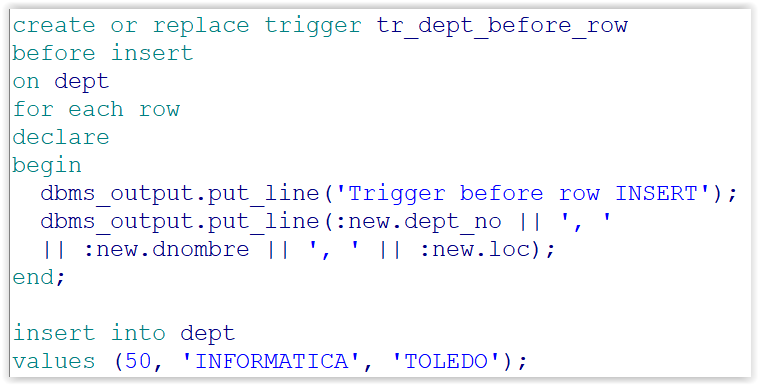
:old.dnombre 🡪 INFORMATICA

:old.loc 🡪 TOLEDO

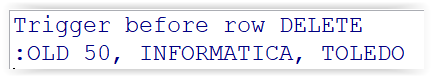
**Ejemplos de variables en Trigger**

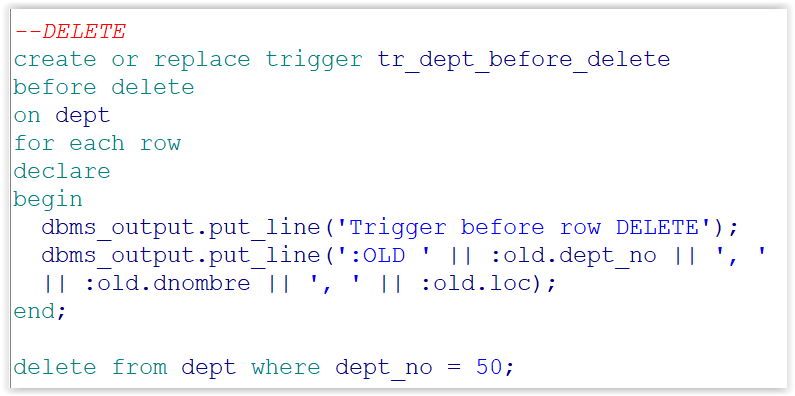
INSERTAR



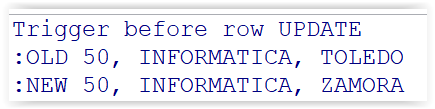


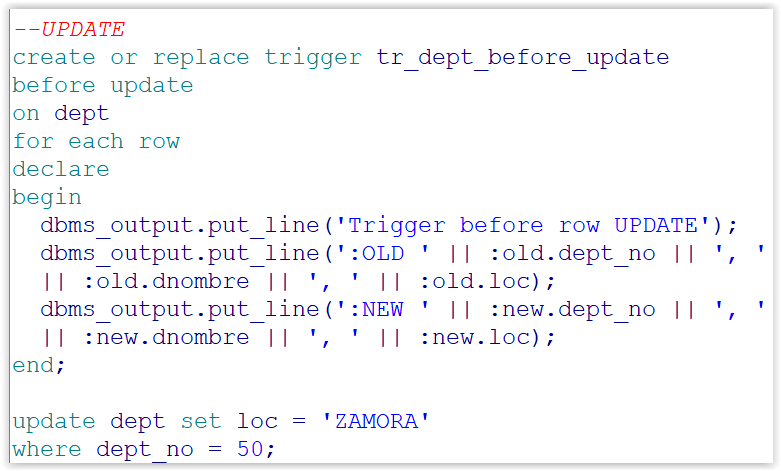
ELIMINAR





UPDATE



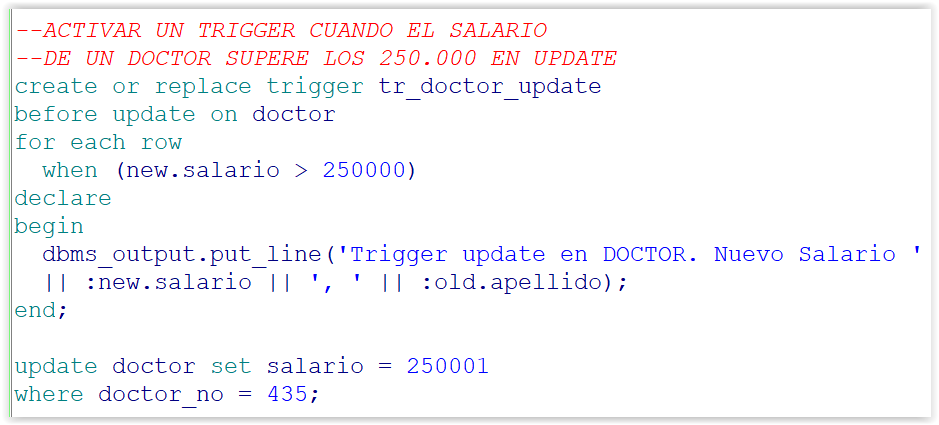


**CLAUSULA WHEN EN TRIGGERS**

Dicha clausula opcional nos permite poder indicar cuando deseamos que se active el Trigger mediante una condición.

Para poder utilizar la cláusula, es necesario tener un Trigger FOR EACH ROW, dónde preguntaremos por algún campo de :new o de :old para la condición de activación.





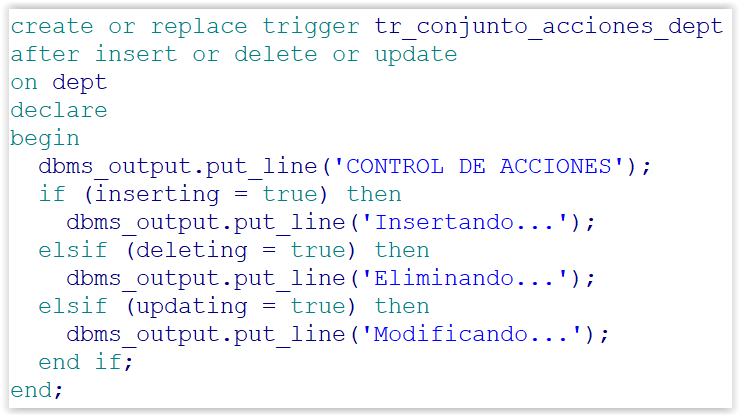
No es habitual, pero podemos crear un trigger que contenga todas las acciones a la

vez.

En estos casos, dentro del Trigger, tenemos unas variables que nos indicarán el tipo

de consulta que estamos realizando.

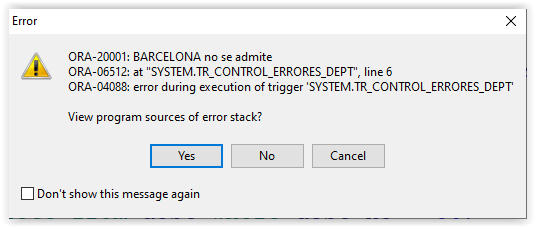
* INSERTING: Consulta insert
* DELETING: Consulta delete
* UPDATING: Consulta update

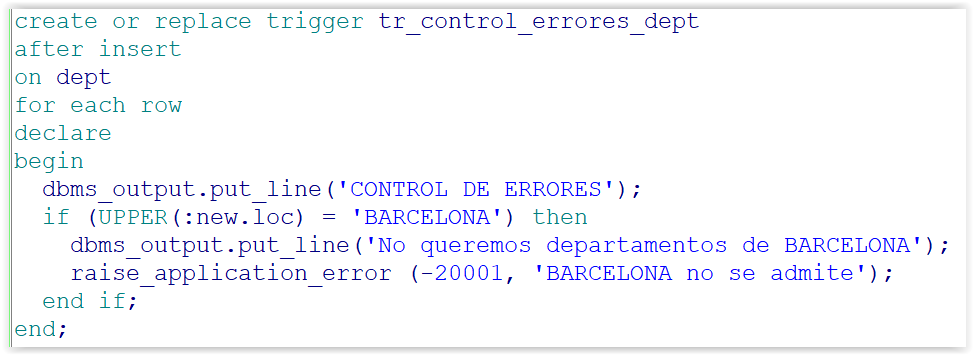


**CONTROL DE ERRORES**

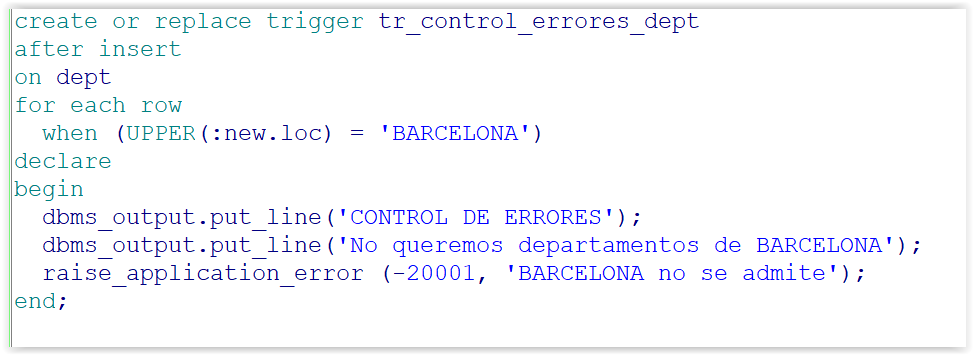
No podemos bloquear acciones por sí mismas dentro de los triggers, a no ser que lancemos errores de aplicación.

Mediante errores, podemos bloquear las acciones para las que han sido creadas los triggers.



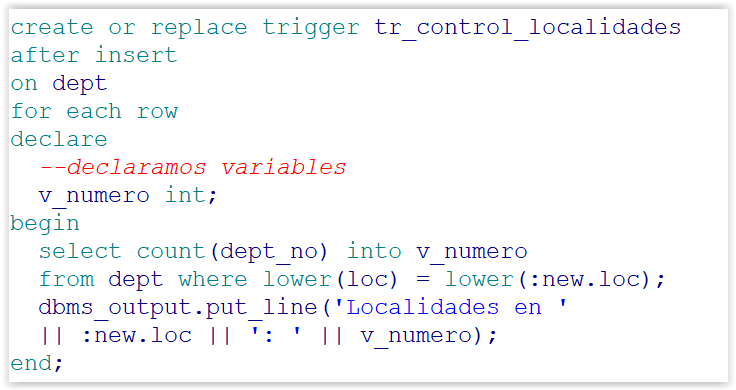


Sintaxis WHEN

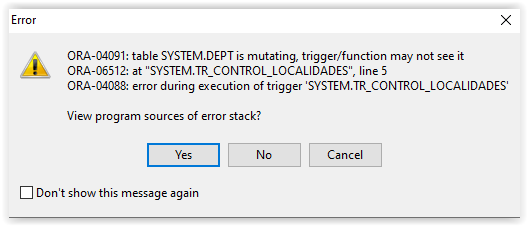


Ejemplo:

Queremos impedir que exista más de un departamento en una localidad. Realizar un trigger para impedir la inserción.

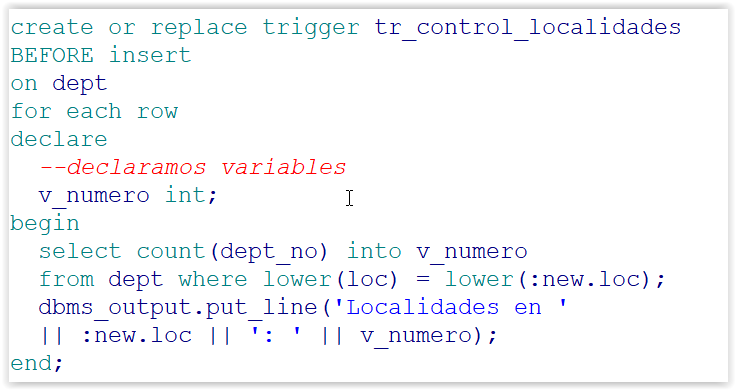


Cuando accedemos a los datos de la tabla, nos indica que está **mutando**.

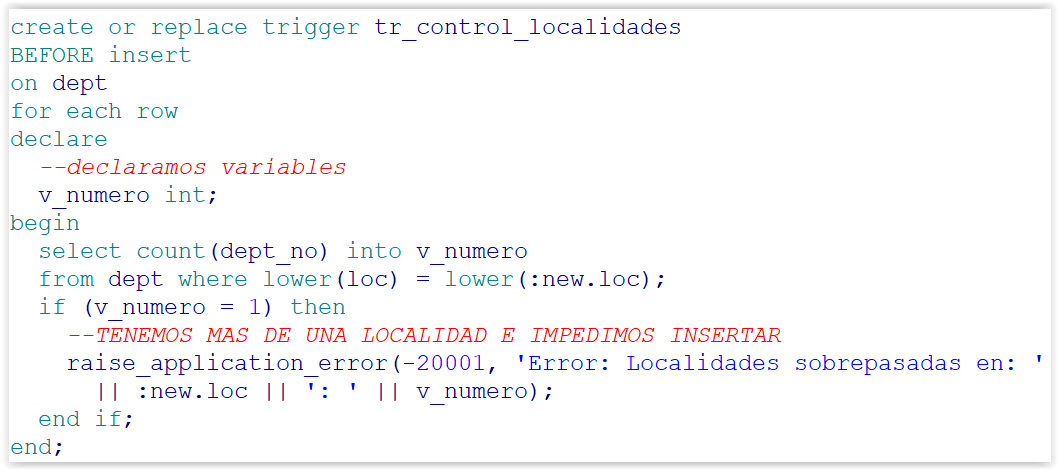


Como la tabla está cambiando (INSERT, UPDATE, DELETE), no podemos realizar consultas sobre la propia tabla mientras está realizando las acciones en **AFTER**.

Si queremos consultar datos de la tabla, debemos hacer las consultas dentro de un trigger **BEFORE en INSERT**.



Solución del ejemplo:



**INTEGRIDAD RELACIONAL**

Los Triggers también se utilizan para mantener la integridad relacional entre las tablas.

Como todos sabemos, con una Foreign Key se mantiene la integridad relacional en las tablas.

Lógicamente, si utilizamos un Trigger para integridad relacional, impediríamos la acción de cambiar el ID con un raise\_application\_error.

Por ejemplo, si intentamos **eliminar** un dato de una tabla principal (Clientes) y tiene una tabla con datos relacionados (Pedidos), nos impedirá realizar la acción porque existe integridad relacional entre los datos.

Podemos modificar el Foreign Key e indicar que, si eliminamos un departamento, elimine a su vez todos los datos relacionados, es decir, los empleados.

ALTER TABLE EMP

ADD CONSTRAINT FK\_DEPT\_EMP

FOREIGN KEY (DEPT\_NO)

REFERENCES DEPT (DEPT\_NO)

**ON DELETE CASCADE;**

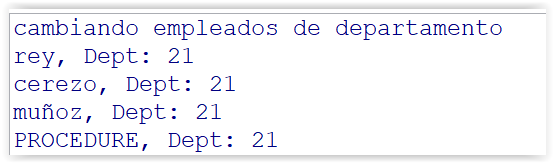
Lo que no existe en Oracle es la integridad referencial en el caso de que realicemos un UPDATE.

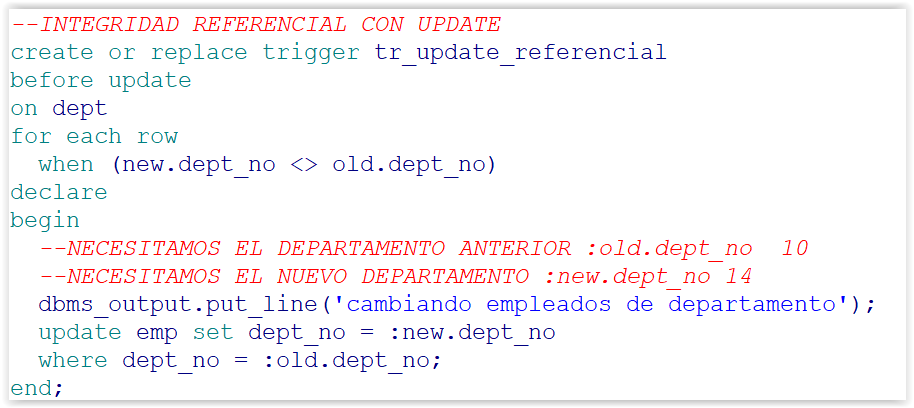
No es habitual realizar el cambio de IDs.

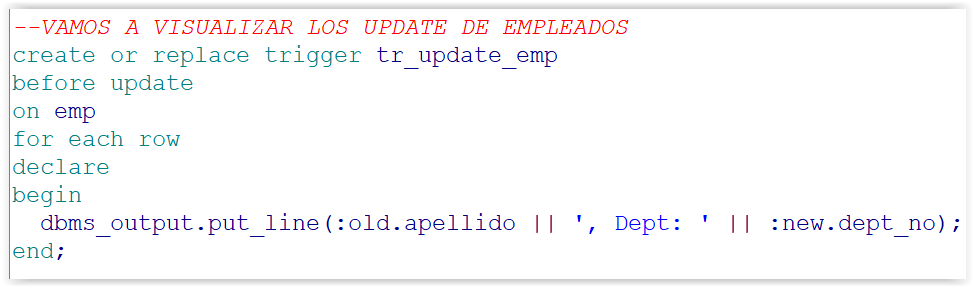
Si modificamos el ID de un cliente, no se modifican sus pedidos al nuevo ID.

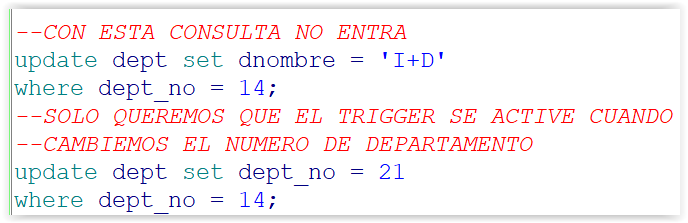
Si deseamos realizar la integridad referencial con UPDATE, debemos utilizar Triggers.

Ejemplo con un Trigger dónde cambiaremos el ID de un departamento y debemos modificar todos los datos asociados (empleados).



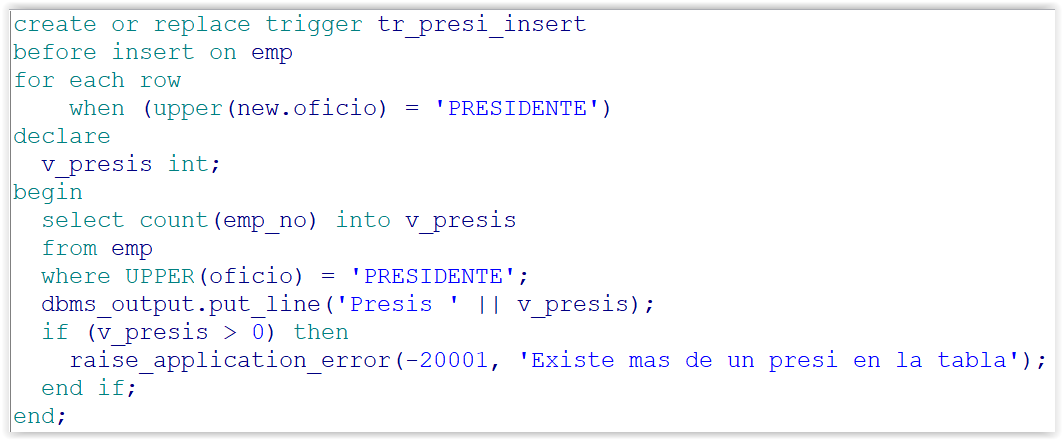


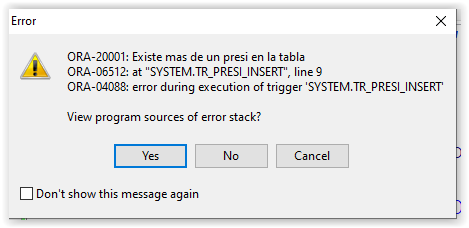




Ejemplo:

Impedir insertar un nuevo PRESIDENTE si ya existe uno en la tabla EMP.





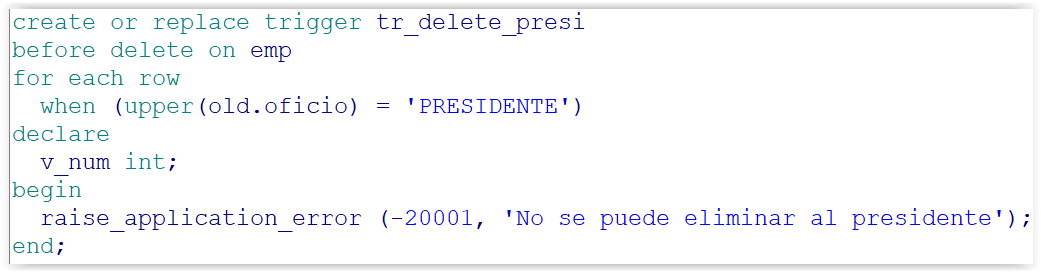
*--COMPROBAR TRIGGER*

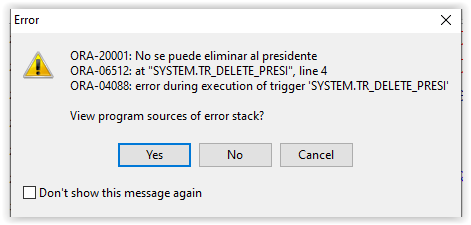
insert into emp values

(1114, 'REINA', 'PRESIDENTE', null, sysdate, 750000, 0, 10);

**Ejemplo:**

Impedir eliminar al PRESIDENTE de los empleados.



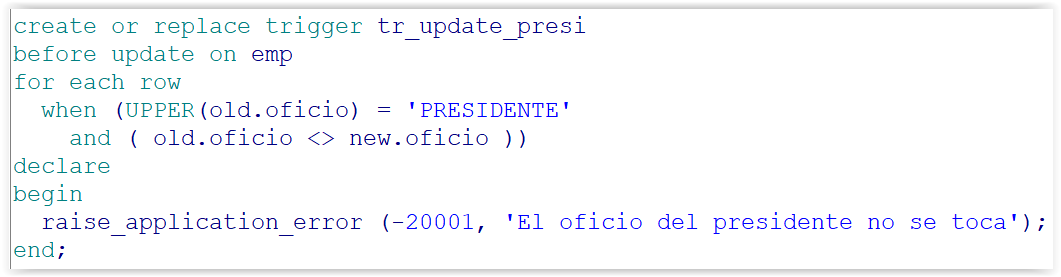


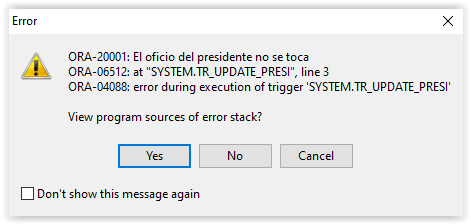
--COMPROBAR EL TRIGGER

delete from emp where emp\_no = 7839;

**Ejemplo:**

Impedir modificar el oficio del PRESIDENTE





--COMPROBAR EL TRIGGER

update emp set salario = salario + 1000

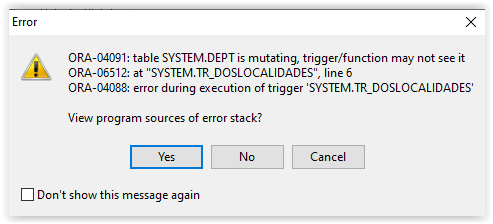
where emp\_no = 7839;

update emp set oficio = 'MINDUNDI'

where emp\_no = 7839;

**SOLUCION A TRIGGER MUTANTES CUANDO QUEREMOS INFORMACION DE LA TABLA**

Dentro de triggers de tipo **FOR EACH ROW** que sean UPDATE o DELETE, no podemos consultar los datos de las tablas sobre las que estemos trabajando.



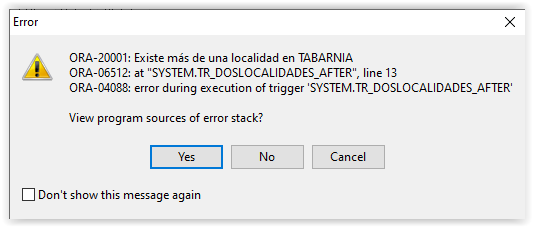
Si no fueran FOR EACH ROW, podríamos consultar perfectamente, pero NO tenemos acceso a las variables **:new** y **:old**.

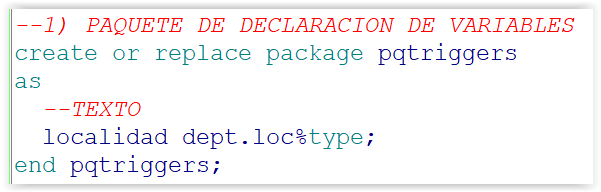
La solución para estos casos es utilizar paquetes de declaración de variables.

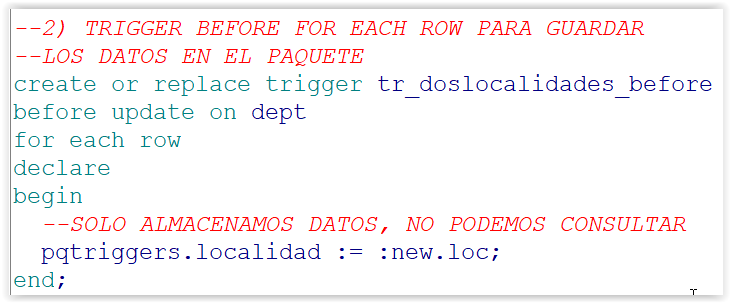
1. Declarar en un paquete los datos que deseamos almacenar de :new o de :old.
2. Utilizar un trigger BEFORE FOR EACH ROW para guardar los datos en el paquete.
3. Utilizar un trigger AFTER para realizar las consultas sobre las variables declaradas en el paquete.

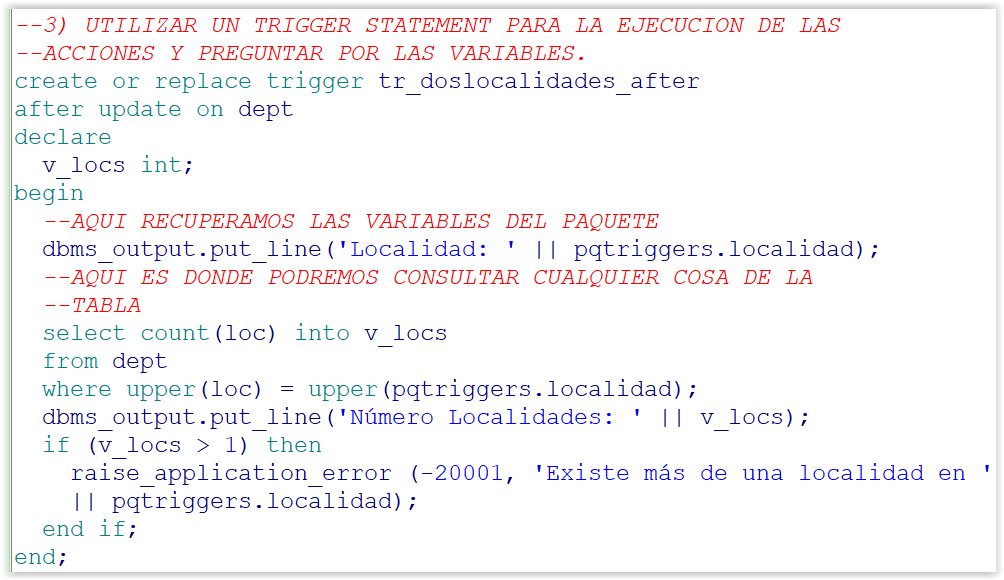
**Ejemplo:**

No podemos tener más de una localidad en un departamento. Realizar un trigger con Update.









**CODIGO SQL**

--1) PAQUETE DE DECLARACION DE VARIABLES

create or replace package pqtriggers

as

--TEXTO

localidad dept.loc%type;

end pqtriggers;

--2) TRIGGER BEFORE FOR EACH ROW PARA GUARDAR

--LOS DATOS EN EL PAQUETE

create or replace trigger tr\_doslocalidades\_before

before update on dept

for each row

declare

begin

--SOLO ALMACENAMOS DATOS, NO PODEMOS CONSULTAR

pqtriggers.localidad := :new.loc;

end;

--3) UTILIZAR UN TRIGGER STATEMENT PARA LA EJECUCION DE LAS

--ACCIONES Y PREGUNTAR POR LAS VARIABLES.

create or replace trigger tr\_doslocalidades\_after

after update on dept

declare

v\_locs int;

begin

--AQUI RECUPERAMOS LAS VARIABLES DEL PAQUETE

dbms\_output.put\_line('Localidad: ' || pqtriggers.localidad);

--AQUI ES DONDE PODREMOS CONSULTAR CUALQUIER COSA DE LA

--TABLA

select count(loc) into v\_locs

from dept

where upper(loc) = upper(pqtriggers.localidad);

dbms\_output.put\_line('Número Localidades: ' || v\_locs);

if (v\_locs > 1) then

raise\_application\_error (-20001, 'Existe más de una localidad en '

|| pqtriggers.localidad);

end if;

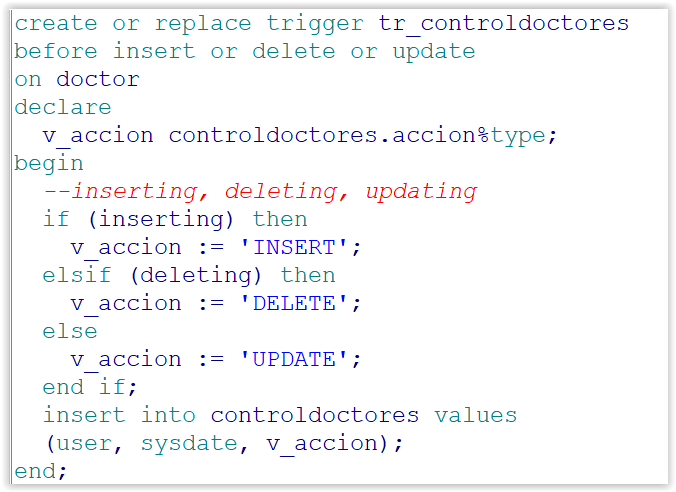
end;

**Ejemplo:**

Realizar un trigger dónde almacenaremos las acciones realizadas sobre la tabla doctor.

Debemos crearnos una tabla llamada **controldoctor**.

Los datos que almacenaremos serán el Usuario, la fecha de la acción y el tipo de acción: INSERT, DELETE, UPDATE.



create or replace trigger tr\_controldoctores

before insert or delete or update

on doctor

declare

v\_accion controldoctores.accion%type;

begin

--inserting, deleting, updating

if (inserting) then

v\_accion := 'INSERT';

elsif (deleting) then

v\_accion := 'DELETE';

else

v\_accion := 'UPDATE';

end if;

insert into controldoctores values

(user, sysdate, v\_accion);

end;

**--PROBAMOS EL TRIGGER**

insert into doctor values (19, 444, 'House G.', 'Diagnostico', 500000);

update doctor set salario = salario + 200

where doctor\_no = 444;

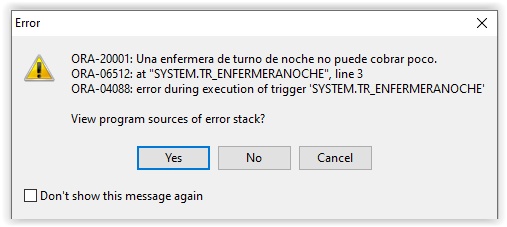
delete from doctor where doctor\_no = 444;

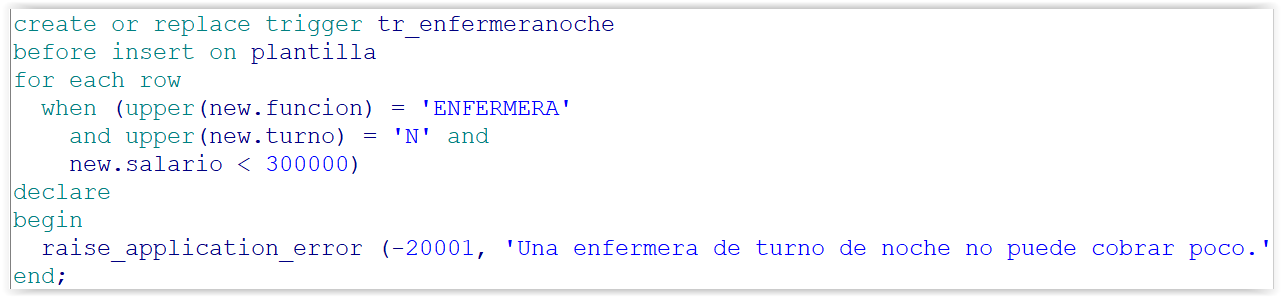
**Ejemplo**

Realizar un Trigger que comprobará el salario de una enfermera del turno de noche.

Comprobaremos que una enfermera del turno de noche cobre siempre por encima de 300.000.

Realizar un trigger insert para comprobar la acción.





create or replace trigger tr\_enfermeranoche

before insert on plantilla

for each row

when (upper(new.funcion) = 'ENFERMERA'

and upper(new.turno) = 'N' and

new.salario < 300000)

declare

begin

raise\_application\_error (-20001, 'Una enfermera de turno de noche no puede cobrar poco.');

end;

--COMPROBAR TRIGGER

insert into plantilla values

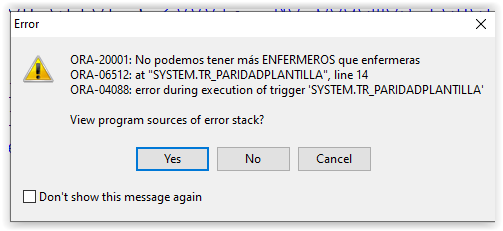
(22, 3, 2, 'EnfermeraOK', 'Enfermera', 'T', 200000);

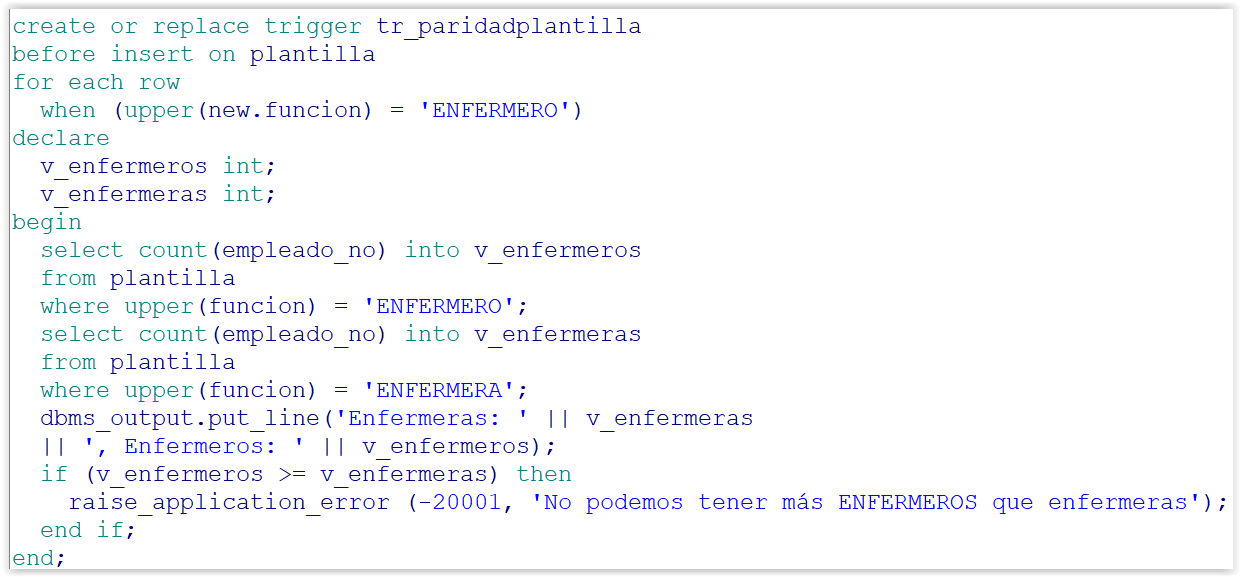
**Ejemplo**

Realizar un Trigger para comprobar la paridad entre enfermeros y enfermeras.

No podremos tener más enfermeros que enfermeras.

Impedir la acción de insertar nuevos enfermeros si se rompe la paridad.





**CODIGO SQL**

create or replace trigger tr\_paridadplantilla

before insert on plantilla

for each row

when (upper(new.funcion) = 'ENFERMERO')

declare

v\_enfermeros int;

v\_enfermeras int;

begin

select count(empleado\_no) into v\_enfermeros

from plantilla

where upper(funcion) = 'ENFERMERO';

select count(empleado\_no) into v\_enfermeras

from plantilla

where upper(funcion) = 'ENFERMERA';

dbms\_output.put\_line('Enfermeras: ' || v\_enfermeras

|| ', Enfermeros: ' || v\_enfermeros);

if (v\_enfermeros >= v\_enfermeras) then

raise\_application\_error (-20001, 'No podemos tener más ENFERMEROS que enfermeras');

end if;

end;

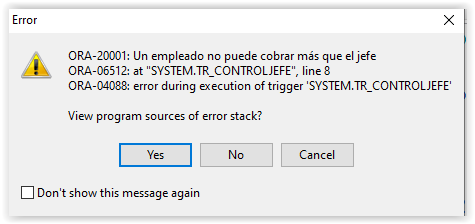
**Ejemplo:**

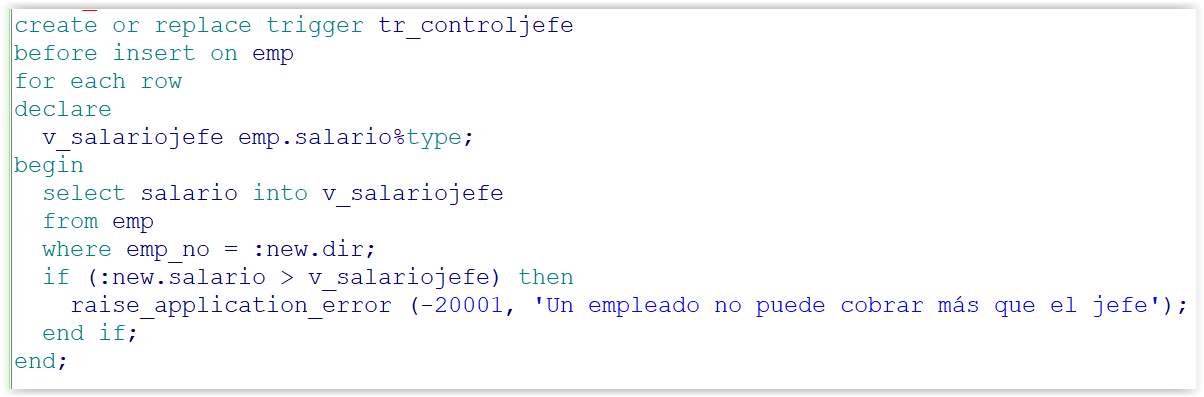
Realizar un trigger para comprobar el salario de los empleados.

Cuando insertemos un nuevo empleado, debemos comprobar que el

empleado insertado no cobra más que su jefe.

EMP\_NO, DIR (DIRECTOR)





**CODIGO SQL**

create or replace trigger tr\_controljefe

before insert on emp

for each row

declare

v\_salariojefe emp.salario%type;

begin

select salario into v\_salariojefe

from emp

where emp\_no = :new.dir;

if (:new.salario > v\_salariojefe) then

raise\_application\_error (-20001, 'Un empleado no puede cobrar más que el jefe');

end if;

end;

**TRIGGERS INSTEAD OF**

Son exactamente iguales a los triggers que hemos vistos en los ejemplos anteriores, con la diferencia que estos triggers se ejecutan solamente con consultas de acción en **VISTAS**.

Este tipo de triggers están pensados para solucionar problemas que podemos tener en consultas de acción sobre las vistas, ya que las vistas no contienen toda la información de una tabla, solamente unos campos.

Un trigger instead of impide realizar cambios sobre una tabla.

Nosotros debemos realizar la acción EXPLICITA para la que ha sido creado el Trigger.

Si un Trigger está creado en una vista sobre INSERT, debemos insertar nosotros manualmente dentro del trigger.

Sintaxis:

create or replace trigger nombre\_trigger

instead of acción

on vista

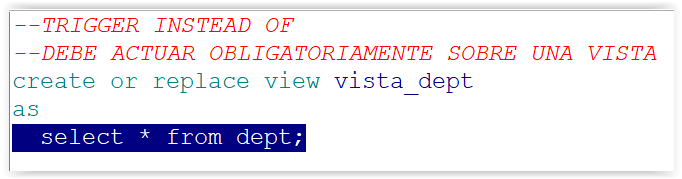
declare

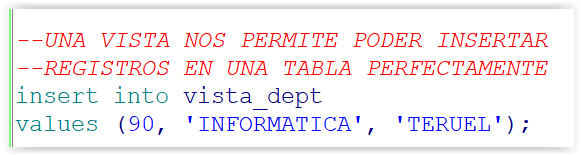
begin

--aquí debemos realizar la acción

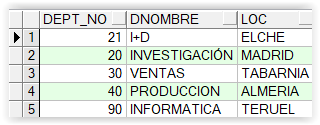
end;

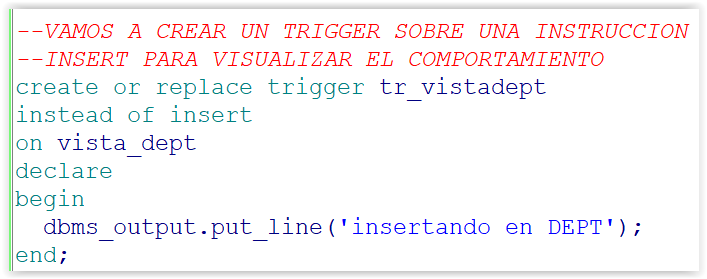
Dentro de un trigger instead of, tendremos acceso a **:old** y a **:new**

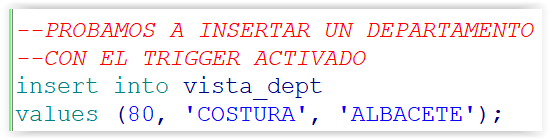




Veremos que ha insertado el registro



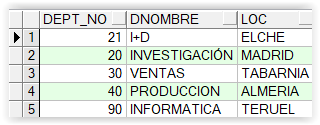


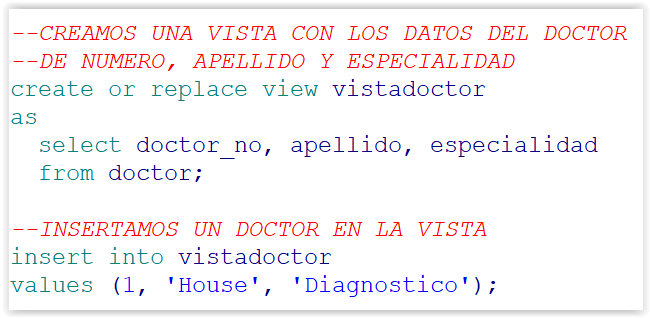


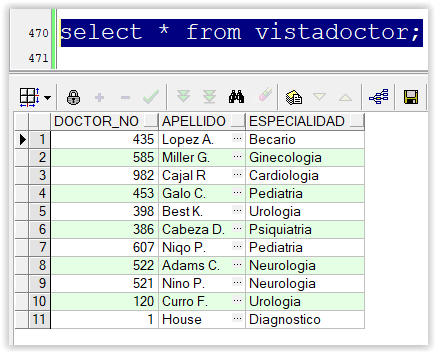
Veremos el mensaje de inserción del TRIGGER



Pero podremos comprobar que no ha insertado en nuestra tabla.







--EL PROGRAMADOR QUE ACABA DE INSERTAR

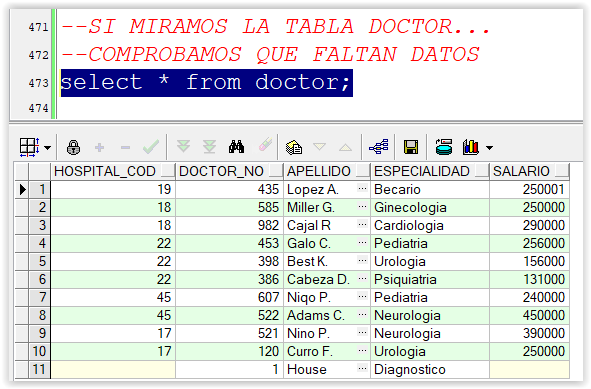
--CON LA VISTA, NO SABE QUE HAY MAS

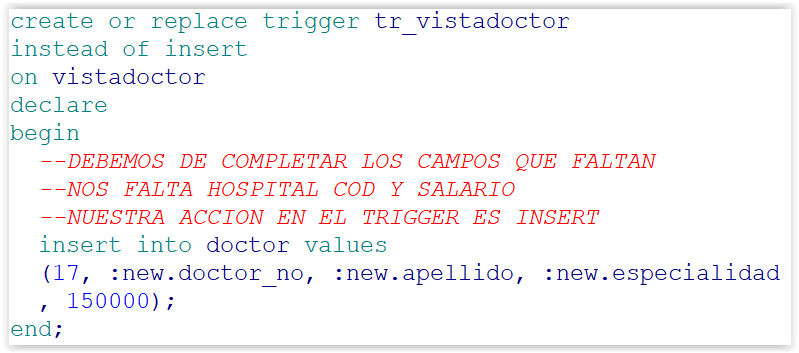
--CAMPOS Y NUNCA PODRA SABERLO (SEGURIDAD)

--ESTE TIPO DE TRIGGERS SE PUEDEN UTILIZAR

--PARA COMPLETAR LOS CAMPOS QUE FALTEN EN

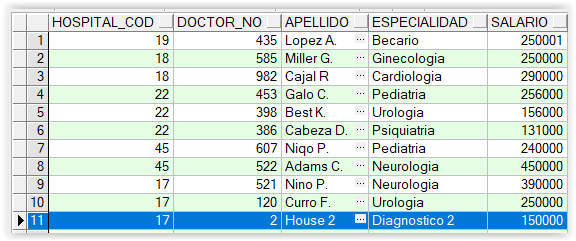
--UNA TABLA





insert into vistadoctor

values (2, 'House 2', 'Diagnostico 2');



**Ejemplo**

--COMO SI NO EXISTIERAN HOSPITAL COD NI SALA COD

--QUEREMOS UNA VISTA QUE NOS MUESTRE

--EL ID DEL EMPLEADO, APELLIDO, FUNCION Y TURNO

--DE LA PLANTILLA

--EL SALARIO ES UN DATO SENSIBLE Y NO QUEREMOS

--VISUALIZARLO.

--SI INSERTAN UN EMPLEADO SOBRE LA VISTA

--, NO QUEREMOS DEJAR VACIO EL CAMPO SALARIO.

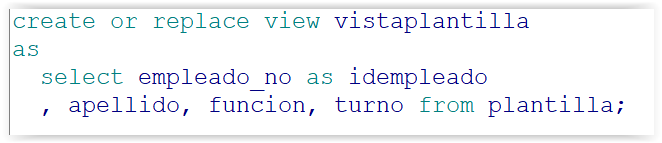
--TENEMOS UNA HERRAMIENTA LLAMADA DEFAULT

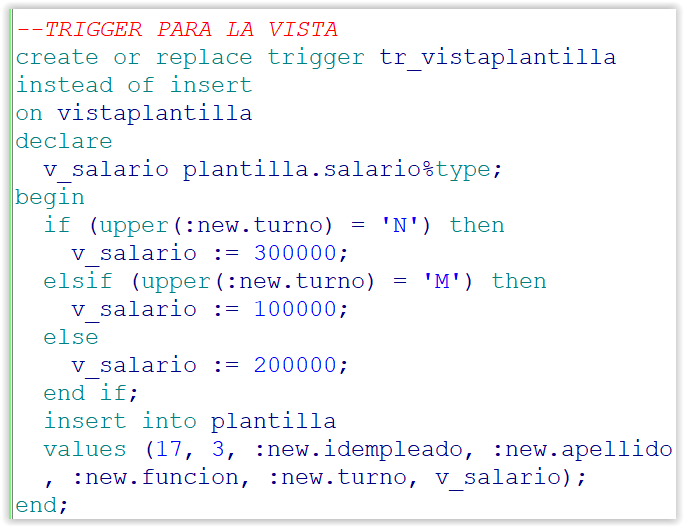
--QUEREMOS DAR UN SALARIO BASE A CADA PERSONA POR SU TURNO

--NOCHE: 300000

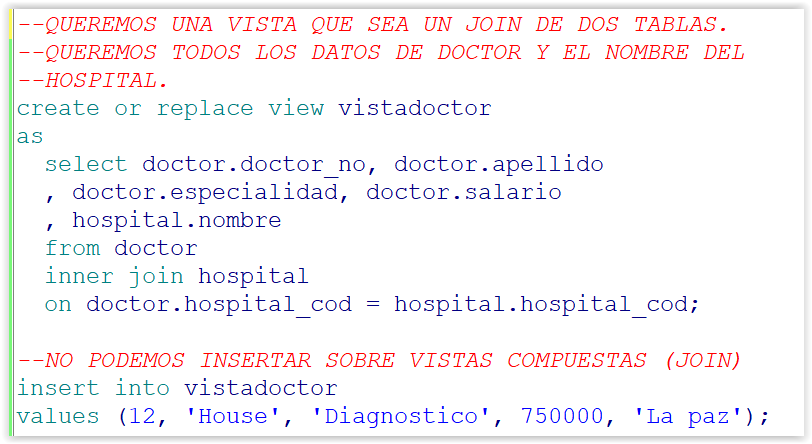
--TARDE: 200000

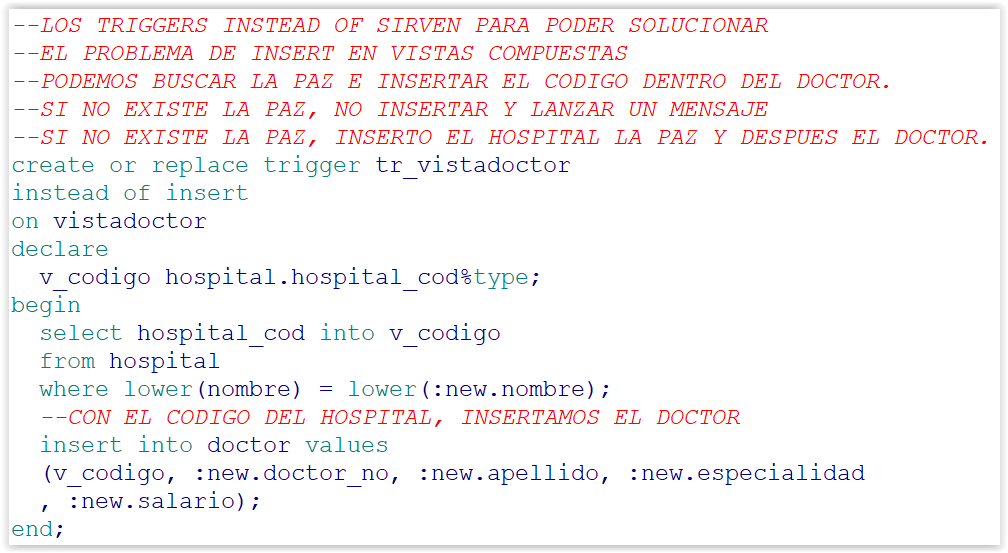
--MAÑANA: 100000



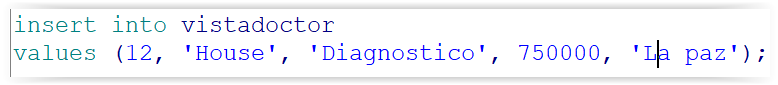


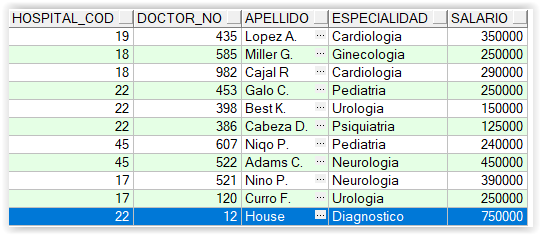
**Utilización de este tipo de Triggers con varias tablas.**





Ya podremos realizar la inserción de datos.





**CODIGO SQL**

--QUEREMOS UNA VISTA QUE SEA UN JOIN DE DOS TABLAS.

--QUEREMOS TODOS LOS DATOS DE DOCTOR Y EL NOMBRE DEL

--HOSPITAL.

create or replace view vistadoctor

as

select doctor.doctor\_no, doctor.apellido

, doctor.especialidad, doctor.salario

, hospital.nombre

from doctor

inner join hospital

on doctor.hospital\_cod = hospital.hospital\_cod;

**--NO PODEMOS INSERTAR SOBRE VISTAS COMPUESTAS (JOIN)**

insert into vistadoctor

values (12, 'House', 'Diagnostico', 750000, 'La paz');

--LOS TRIGGERS INSTEAD OF SIRVEN PARA PODER SOLUCIONAR

--EL PROBLEMA DE INSERT EN VISTAS COMPUESTAS

--PODEMOS BUSCAR LA PAZ E INSERTAR EL CODIGO DENTRO DEL DOCTOR.

--SI NO EXISTE LA PAZ, NO INSERTAR Y LANZAR UN MENSAJE

--SI NO EXISTE LA PAZ, INSERTO EL HOSPITAL LA PAZ Y DESPUES EL DOCTOR.

create or replace trigger tr\_vistadoctor

instead of insert

on vistadoctor

declare

v\_codigo hospital.hospital\_cod%type;

begin

select hospital\_cod into v\_codigo

from hospital

where lower(nombre) = lower(:new.nombre);

--CON EL CODIGO DEL HOSPITAL, INSERTAMOS EL DOCTOR

insert into doctor values

(v\_codigo, :new.doctor\_no, :new.apellido, :new.especialidad

, :new.salario);

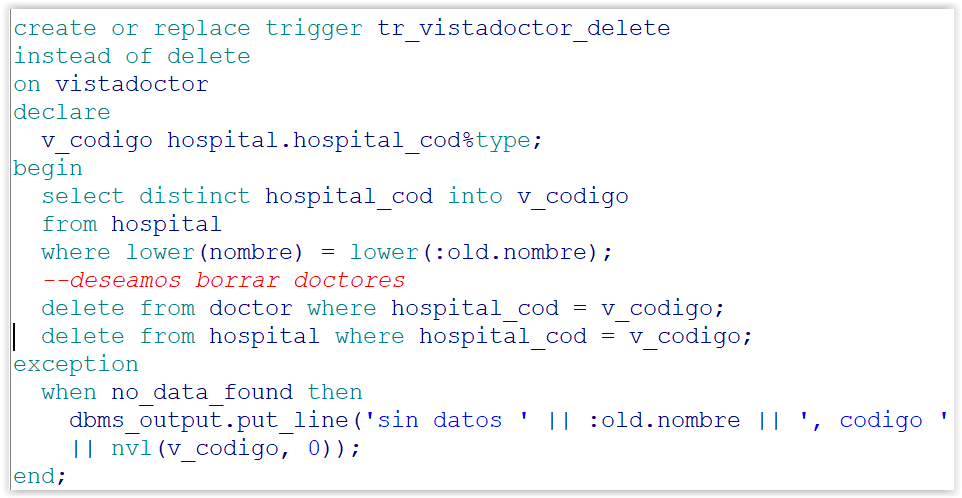
end;

**Ejemplo**

--realizar un trigger que permite eliminar de VISTADOCTOR

--si nos indican el nombre del hospital.

--eliminamos doctores.



delete from vistadoctor where lower(nombre) = lower('general');

**Ejemplo**

--CREAR UNA VISTA CON LOS DATOS DE LOS EMPLEADOS

--EXCEPTO EL SALARIO Y LA COMISION.

--EL SALARIO IRA POR CATEGORIA (OFICIO)

--DIRECTOR: 350000

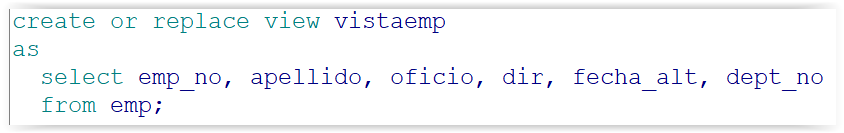
--ANALISTA: 250000

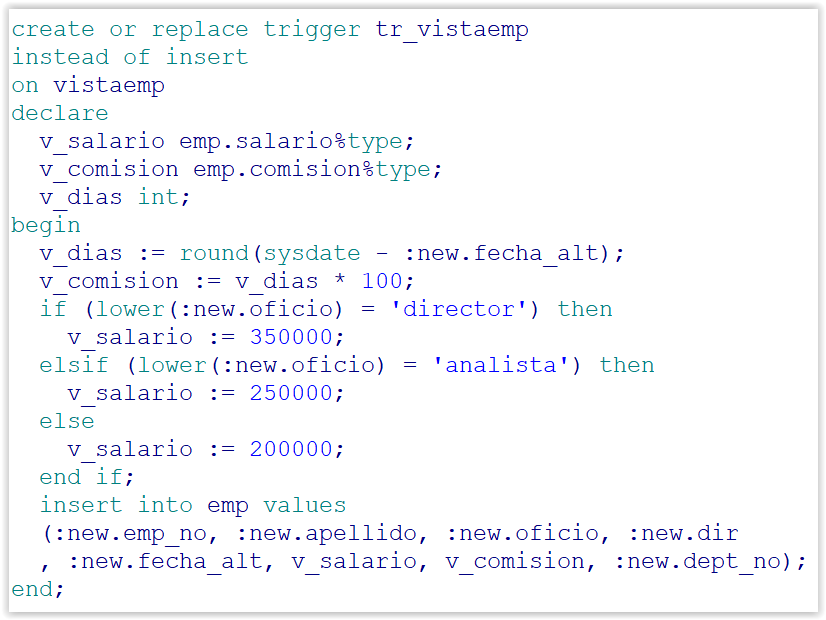
--CUALQUIER OTRO: 200000

--LA COMISION SERA EL NUMERO DE DIAS QUE LLEVA EN LA

--EMPRESA POR 100 36500

--FUNCIONES????!?!?!?





**CODIGO SQL**

create or replace view vistaemp

as

select emp\_no, apellido, oficio, dir, fecha\_alt, dept\_no

from emp;

create or replace trigger tr\_vistaemp

instead of insert

on vistaemp

declare

v\_salario emp.salario%type;

v\_comision emp.comision%type;

v\_dias int;

begin

v\_dias := round(sysdate - :new.fecha\_alt);

v\_comision := v\_dias \* 100;

if (lower(:new.oficio) = 'director') then

v\_salario := 350000;

elsif (lower(:new.oficio) = 'analista') then

v\_salario := 250000;

else

v\_salario := 200000;

end if;

insert into emp values

(:new.emp\_no, :new.apellido, :new.oficio, :new.dir

, :new.fecha\_alt, v\_salario, v\_comision, :new.dept\_no);

end;

insert into vistaemp

values (1115, 'DIRE', 'DIRECTOR', 7698, '01/01/2020', 10);