BD06.

UT06 Programación de bases de datos.

Profesor: Valentín Rebollada Casado.

Alumna: Cherry Reynoso Catalán

DAM Distancia. Base de datos

31/03/2024

Cherry Reynoso Catalán

ÍNDICE

Cherry Reynoso Catalán

Para la realización de la tarea de esta unidad nos basaremos en el caso de estudio expuesto en los contenidos de la misma en el Anexo I. La tarea que te pedimos que realices consta de 2 actividades:

Actividad 1.

Crear **un procedimiento** que *permita cambiar a todos los agentes de una familia determinada (*familia origen) *a otra familia* (familia destino).
El procedimiento tendrá la siguiente cabecera `CambiarAgentesFamilia( id_FamiliaOrigen, id_FamiliaDestino)`, donde cada uno de los argumentos corresponde a un identificador de Familia. Cambiará la columna Identificador de Familia de todos los agentes, de la tabla `AGENTES`, que pertenecen a la Familia con código `id_FamiliaOrigen` por el código `id_FamiliaDestino`
Previamente comprobará que ambas familias existen y que no son iguales.
Para la comprobación de la existencia de las familias se puede utilizar un cursor variable, o contar el número de filas y en caso de que no exista, se visualizará el mensaje correspondiente mediante una **excepción del tipo RAISE_APPLICATION_ERROR.** También se mostrará un mensaje en caso de que ambos argumentos tengan el mismo valor.

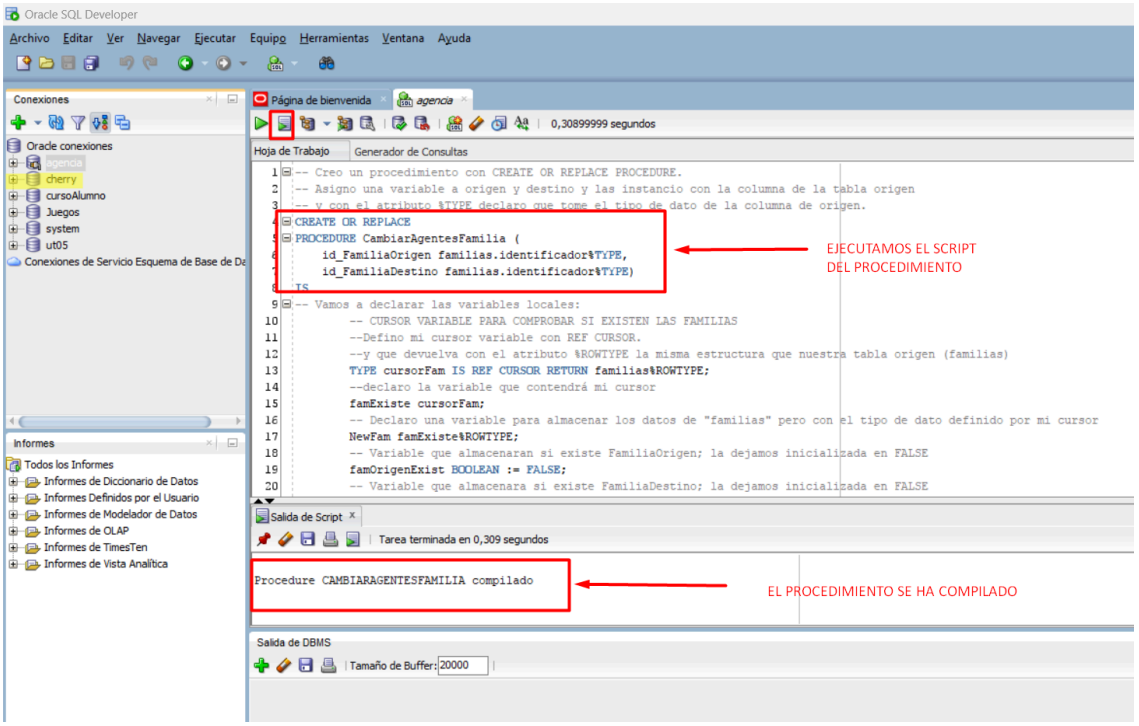
El procedimiento visualizará el mensaje "Se han trasladado XXX agentes de la familia XXXXXX a la familia ZZZZZZ" donde XXX es el número de agentes que se han cambiado de familia, XXXXXX es el nombre de la familia origen y ZZZZZZ es el nombre de la familia destino.

SCRIPT DEL PROCEDIMIENTO USANDO UN CURSOR:

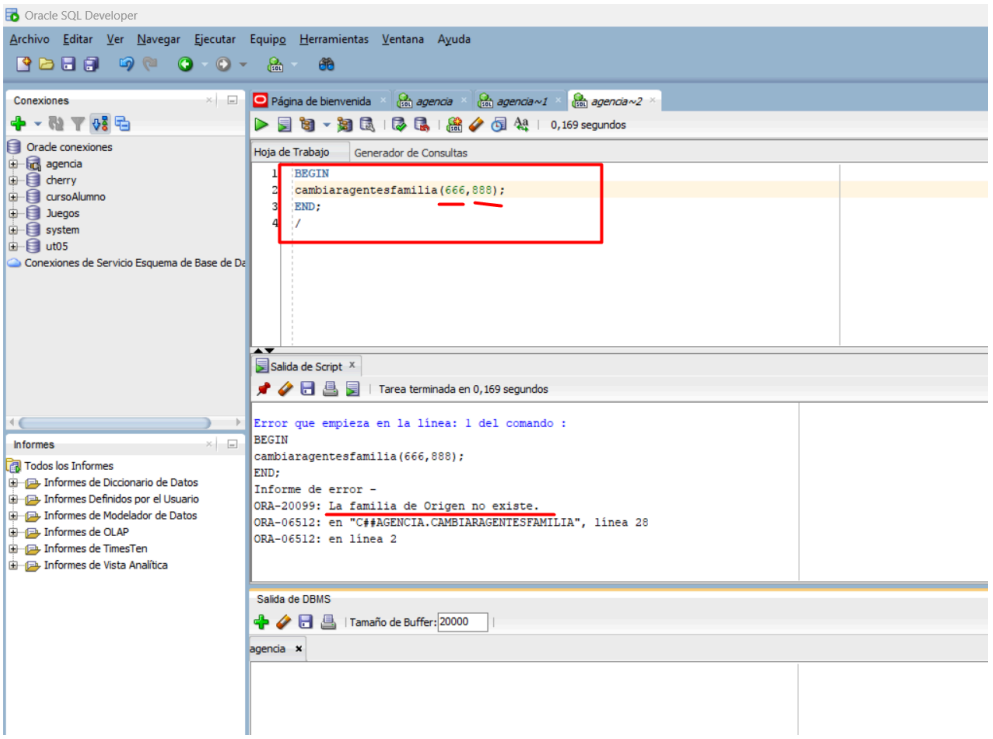
```
-- Creo un procedimiento con CREATE OR REPLACE PROCEDURE.
-- Asigno una variable a origen y destino y las instancio con la
columna de la tabla origen
-- y con el atributo %TYPE declaro que tome el tipo de dato de la
columna de origen.
CREATE OR REPLACE
PROCEDURE CambiarAgentesFamilia (
    id_FamiliaOrigen familias.identificador%TYPE,
    id_FamiliaDestino familias.identificador%TYPE)
IS
-- Vamos a declarar las variables locales:
        -- CURSOR VARIABLE PARA COMPROBAR SI EXISTEN LAS FAMILIAS
        --Defino mi cursor variable con REF CURSOR.
        --y que devuelva con el atributo %ROWTYPE la misma estructura
que nuestra tabla origen (familias)
        TYPE cursorFam IS REF CURSOR RETURN familias%ROWTYPE;
        --declaro la variable que contendrá mi cursor
        famExiste cursorFam;
        -- Declaro una variable para almacenar los datos de "familias"
pero con el tipo de dato definido por mi cursor
        NewFam famExiste%ROWTYPE;
        -- Variable que almacenaran si existe FamiliaOrigen; la dejamos
inicializada en FALSE
        famOrigenExist BOOLEAN := FALSE;
        -- Variable que almacenara si existe FamiliaDestino; la dejamos
inicializada en FALSE
        famDestinoExist BOOLEAN := FALSE;


BEGIN

        --COMPROBAMOS LA EXISTENCIA DE LAS FAMILIAS
        --Una vez definido nuestro cursor variable lo asociamos a una
consulta:
        -- Comprobamos que familia Origen existe
        OPEN famExiste FOR SELECT * FROM familias WHERE identificador=
id_FamiliaOrigen;
        FETCH famExiste INTO NewFam; -- recojo los datos de mi cursor y
los voy almacenando en mi variable NewFam
        IF famExiste%NOTFOUND THEN
```

```
                RAISE_APPLICATION_ERROR(-20099,'La familia de Origen no
existe.');
        ELSE famOrigenExist := TRUE;
        END IF;
        CLOSE famExiste;--Cerramos nuestro cursor

        -- COMPROBAMOS que familia Destino existe:
        OPEN famExiste FOR SELECT * FROM familias WHERE identificador =
id_FamiliaDestino;
        FETCH famExiste INTO NewFam;
        IF famExiste%NOTFOUND THEN
                RAISE_APPLICATION_ERROR(-20100,'La familia de Destino
no existe.');
        ELSE famDestinoExist := TRUE;
        END IF;
        CLOSE famExiste; --Cerramos nuestro cursor

        -- Comprobamos SI AMBAS FAMILIAS son diferentes; si son iguales
se alza un RAISE_APPLICATION_ERROR.
        OPEN famExiste FOR SELECT * FROM familias WHERE identificador =
id_FamiliaOrigen;
        IF id_FamiliaDestino = id_FamiliaOrigen THEN
                RAISE_APPLICATION_ERROR(-20101, 'La familia de Origen
no puede ser la misma que la de Destino.');
        END IF;
        CLOSE famExiste; --Cerramos nuestro cursor

        --ACTUALIZAMOS la tabla AGENTES modificando identificador de
familia con código id_FamiliaOrigin por el de id_FamiliaDestino
        UPDATE Agentes
        SET familia = id_FamiliaDestino
        WHERE familia = id_FamiliaOrigen;



       -- MENSAJE de Confirmación
        DBMS_OUTPUT.PUT_LINE('Se han trasladado ' || SQL%ROWCOUNT || '
agentes de la familia ' || id_FamiliaOrigen || ' a la familia ' ||
id_FamiliaDestino);

END;
/
```

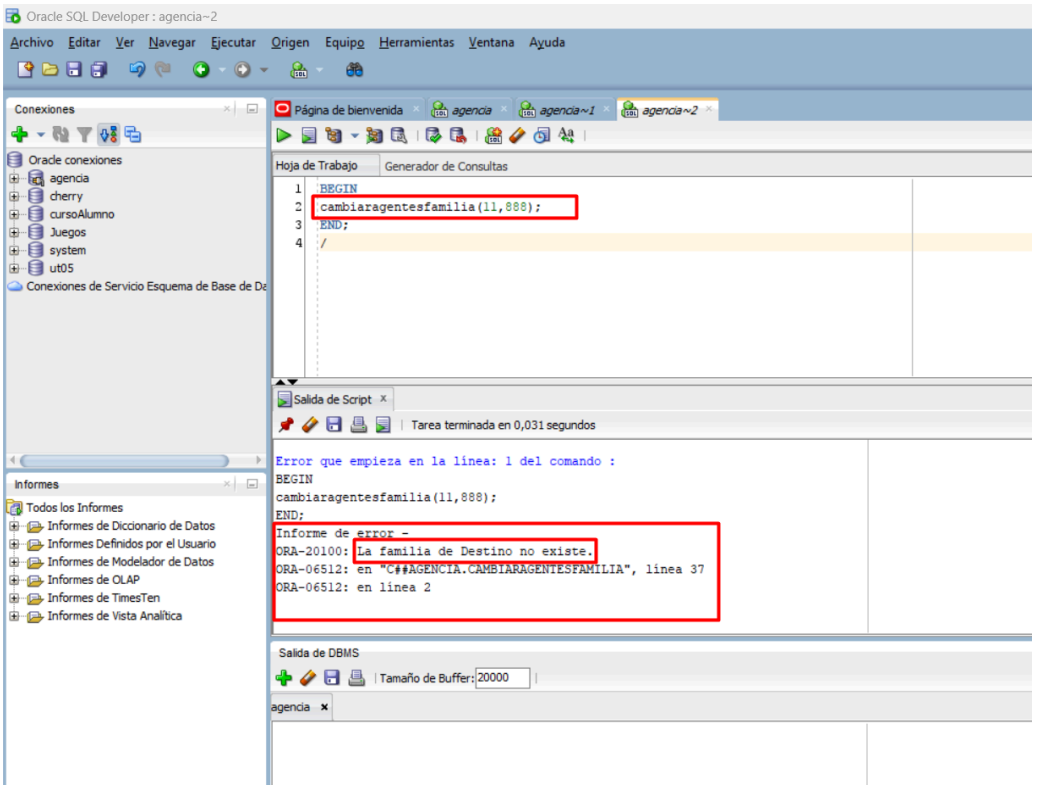
CAPTURAS DE PANTALLAS PARA COMPROBAR EL SCRIPT:



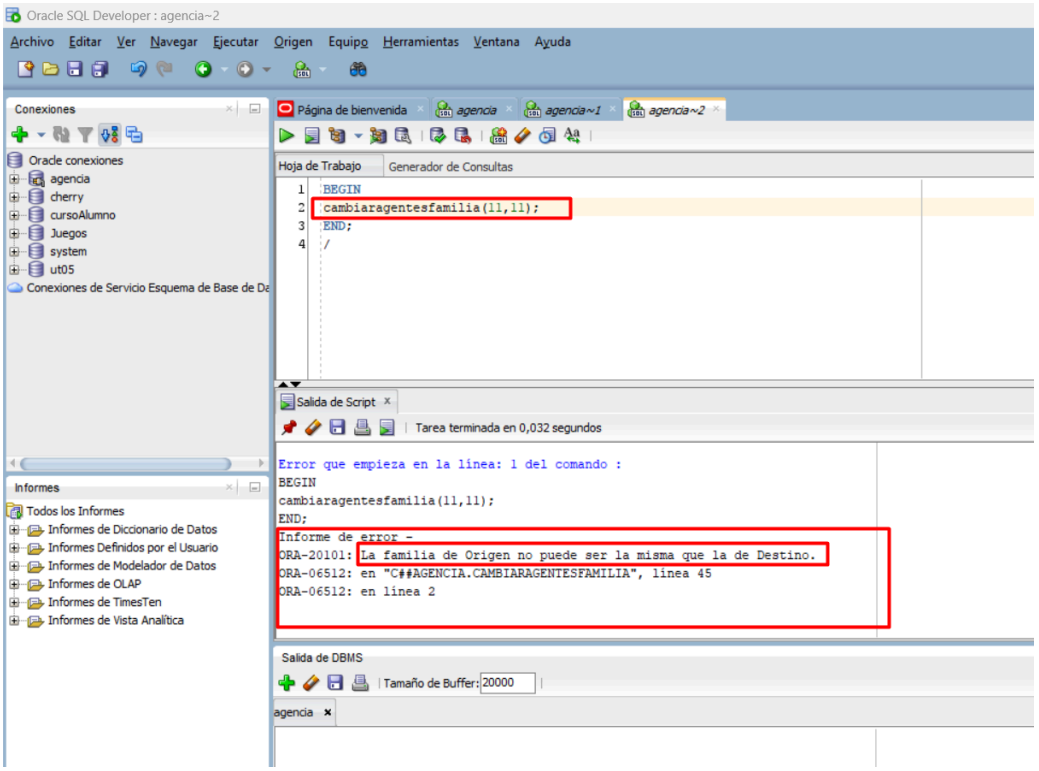
1.1 Ejecutamos el script del procedimiento; el procedimiento se ha compilado correctamente. Cherry Reynoso Catalán.



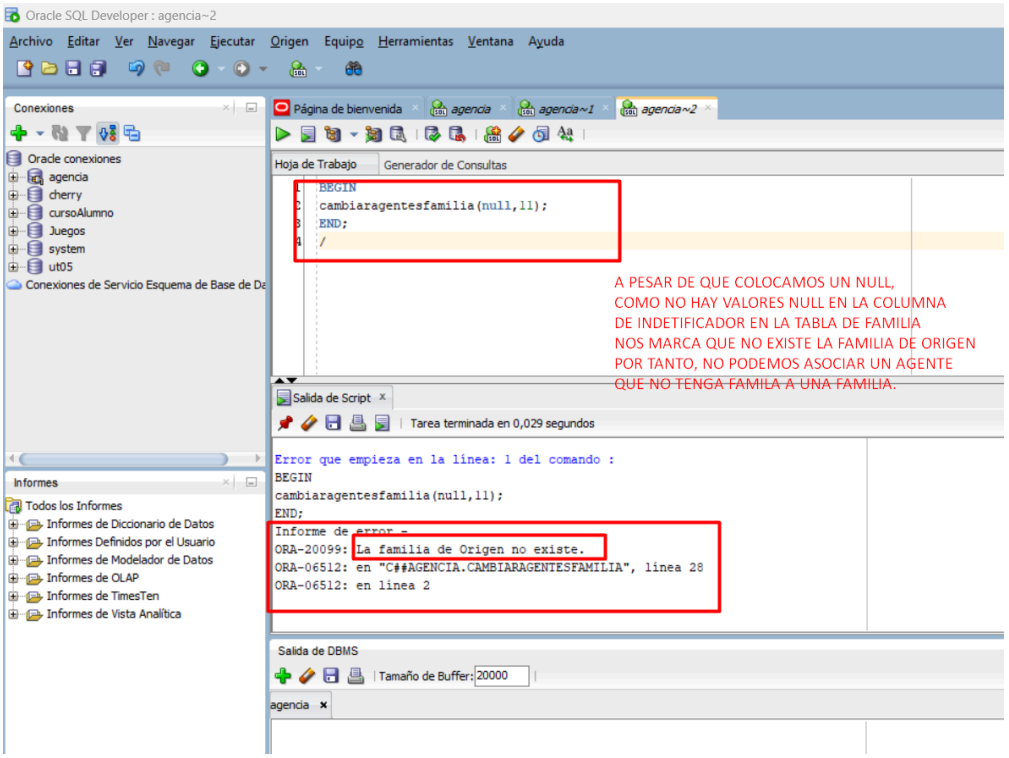
1.2 Probamos el procedimiento con dos familias que no existen; nos salta la primera excepcion. Cherry Reynoso Catalán.



1.3 Probamos el procedimiento con la familia de origen que sí existe pero la de Destino no; nos salta la segunda excepcion. Cherry Reynoso Catalán.



1.4 Probamos el procedimiento ambas familias que existen pero son iguales; nos salta la tercera excepcion. Cherry Reynoso Catalán.



1.5 Probamos el procedimientcon un null; nos salta la primera excepcion debido a que es cronológico(salta la primera excepción que se cumple de las tres). Cherry Reynoso Catalán.

Actividad 2

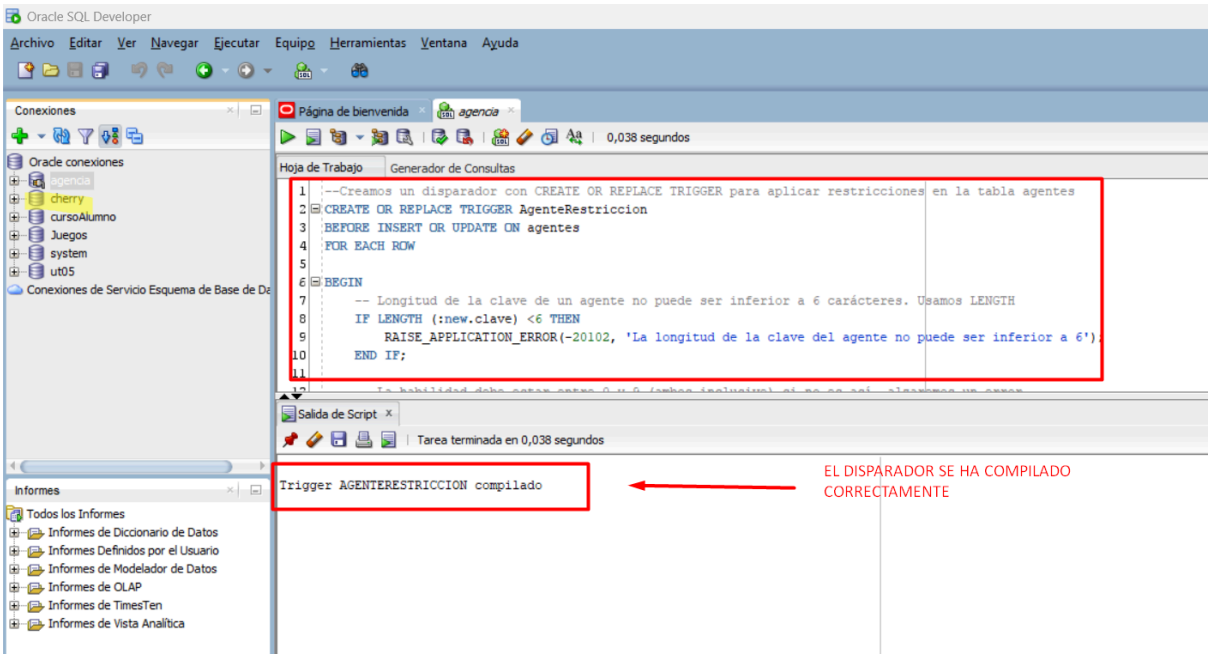
1. Queremos controlar algunas restricciones a la hora de trabajar con agentes:
    a. La longitud de la clave de un agente no puede ser inferior a 6.
    b. La habilidad de un agente debe estar comprendida entre 0 y 9 (ambos inclusive).
    c. La categoría de un agente sólo puede ser igual a 0, 1 o 2.
    d. Si un agente tiene categoría 2 no puede pertenecer a ninguna familia y debe pertenecer a una oficina.
    e. Si un agente tiene categoría 1 no puede pertenecer a ninguna oficina y debe pertenecer a una familia.
    f. Todos los agentes deben pertenecer a una oficina o a una familia pero nunca a ambas a la vez.
2. Se pide crear un disparador para asegurar estas restricciones. El disparador deberá lanzar todos los errores que se puedan producir en su ejecución mediante errores que identifiquen con un mensaje adecuado por qué se ha producido dicho error.
   Algunas de las restricciones implementadas con el disparador se pueden incorporar a la definición del esquema de la tabla utilizando el Lenguaje de Definición de Datos (Check,Unique,..).Identifica cuáles son y con qué tipo de restricciones las implementarías.

SCRIPT DEL DISPARADOR USANDO RAISE_APPLICATION_ERROR:

```
--Creamos un disparador con CREATE OR REPLACE TRIGGER para aplicar
restricciones en la tabla agentes
CREATE OR REPLACE TRIGGER AgenteRestriccion
BEFORE INSERT OR UPDATE ON agentes
FOR EACH ROW

BEGIN
    -- Longitud de la clave de un agente no puede ser inferior a 6
carácteres. Usamos LENGTH
    IF LENGTH (:new.clave) <6 THEN
        RAISE_APPLICATION_ERROR(-20102, 'La longitud de la clave del
agente no puede ser inferior a 6');
    END IF;

    -- La habilidad debe estar entre 0 y 9 (ambos inclusive) si no es
así, alzaremos un error.
    IF :new.habilidad NOT BETWEEN 0 AND 9 THEN
        RAISE_APPLICATION_ERROR(-20103, 'La habilidad debe estar
comprendida entre 0 y 9(ambos inclusive)');
    END IF;
```

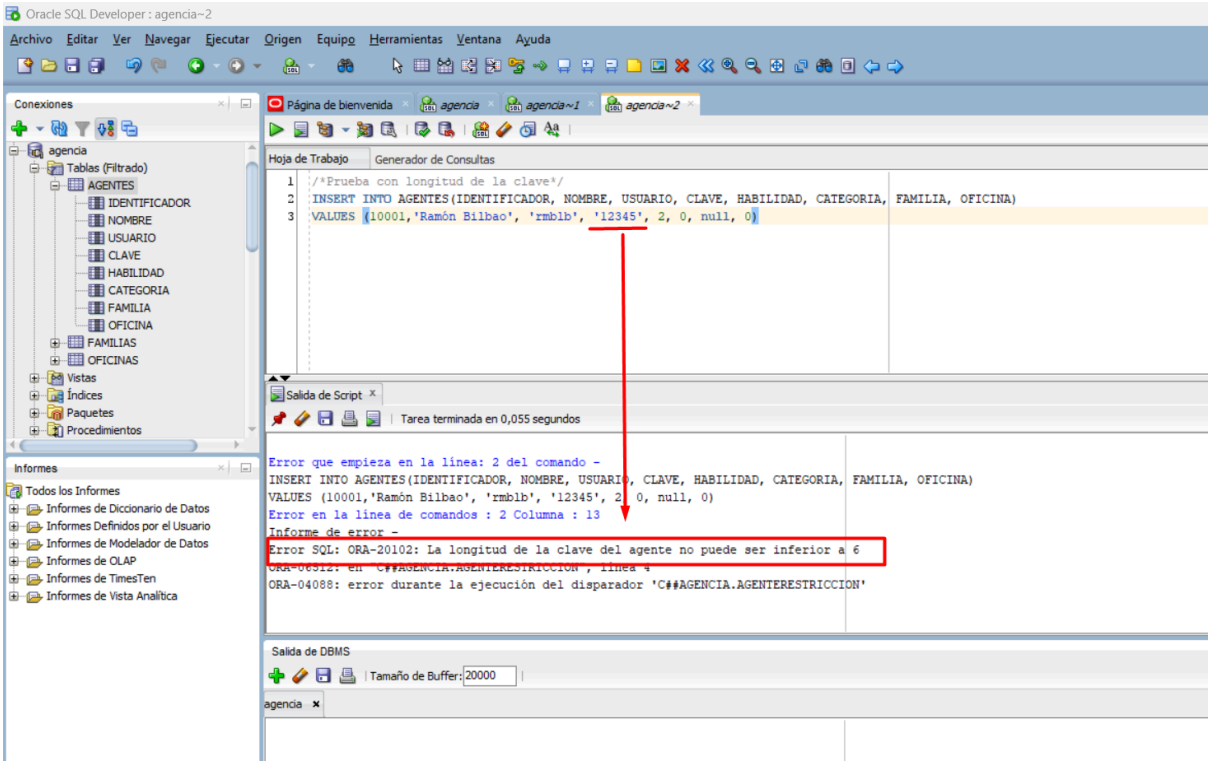
```
    --La categoría de un agente sólo puede ser igual a 0, 1 o 2
    IF :new.categoria NOT IN (0,1,2) THEN
        RAISE_APPLICATION_ERROR(-20104, 'La categoría de un agente sólo
puede ser igual a 0, 1 o 2');
    END IF;

    -- Si un agente tiene categoría 2 no puede pertenecer a ninguna
familia y DEBE pertenecer a una oficina.
    IF (:new.categoria = 2 AND :new.familia IS NOT NULL) OR
(:new.categoria = 2 AND :new.oficina IS NULL) THEN
        RAISE_APPLICATION_ERROR(-20105, 'Un agente con categoría 2 no
puede pertenecer a ninguna familia y debe pertenecer a una oficina.');
    END IF;

    --Si un agente tiene categoría 1 no puede pertenecer a ninguna
oficina y DEBE pertenecer  a una familia
    IF (:new.categoria = 1 AND :new.familia IS NULL) OR (:new.categoria
= 1 AND :new.oficina IS NOT NULL) THEN
        RAISE_APPLICATION_ERROR(-20106, 'Un agente con categoría 1 no
puede pertenecer a ninguna oficina y debe pertenecer a una familia.');
    END IF;

    -- Todos los agentes deben pertenecer  a una oficina o a una
familia pero nunca a ambas a la vez, si no cumple lanza un error.
    IF (:new.familia IS NOT NULL AND :new.oficina IS NOT NULL) OR
(:new.familia IS NULL AND :new.oficina IS NULL) THEN
        RAISE_APPLICATION_ERROR(-20107, 'Todos los agentes deben
pertenecer a una oficina o a una familia pero nunca a ambas a la vez');
    END IF;
END;
/
```

CAPTURAS DE PANTALLAS PARA COMPROBAR EL SCRIPT



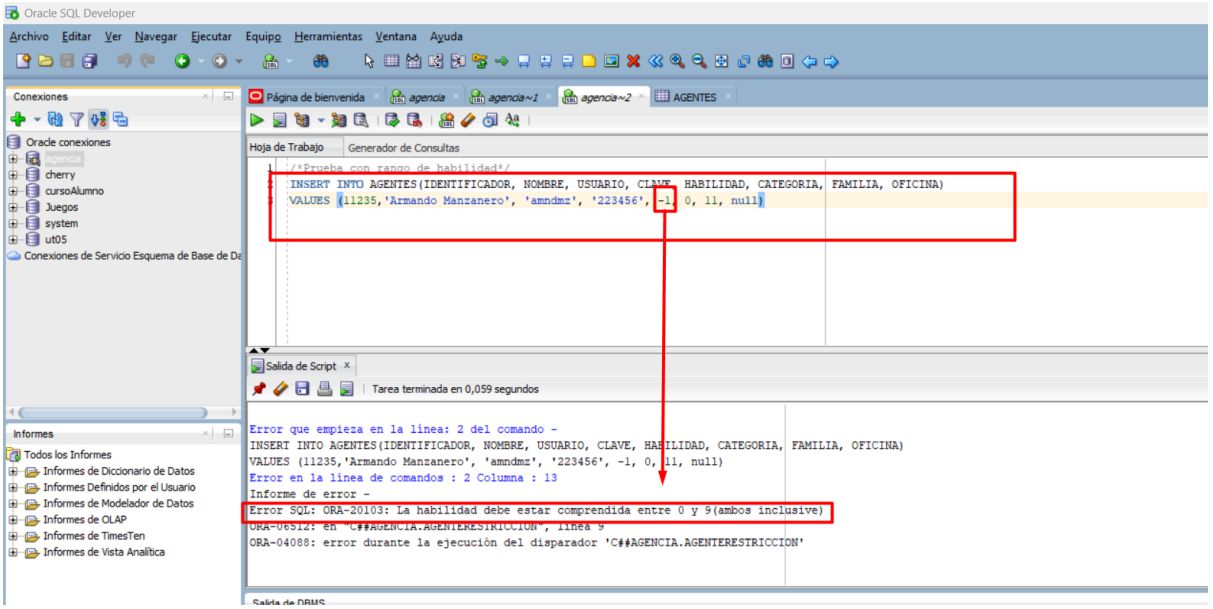
2.1 Ejecutamos nuestro script del “trigger” y vemos que se ha compilado correctamente. Cherry Reynoso Catalán.

1. **Prueba con longitud de la clave:**



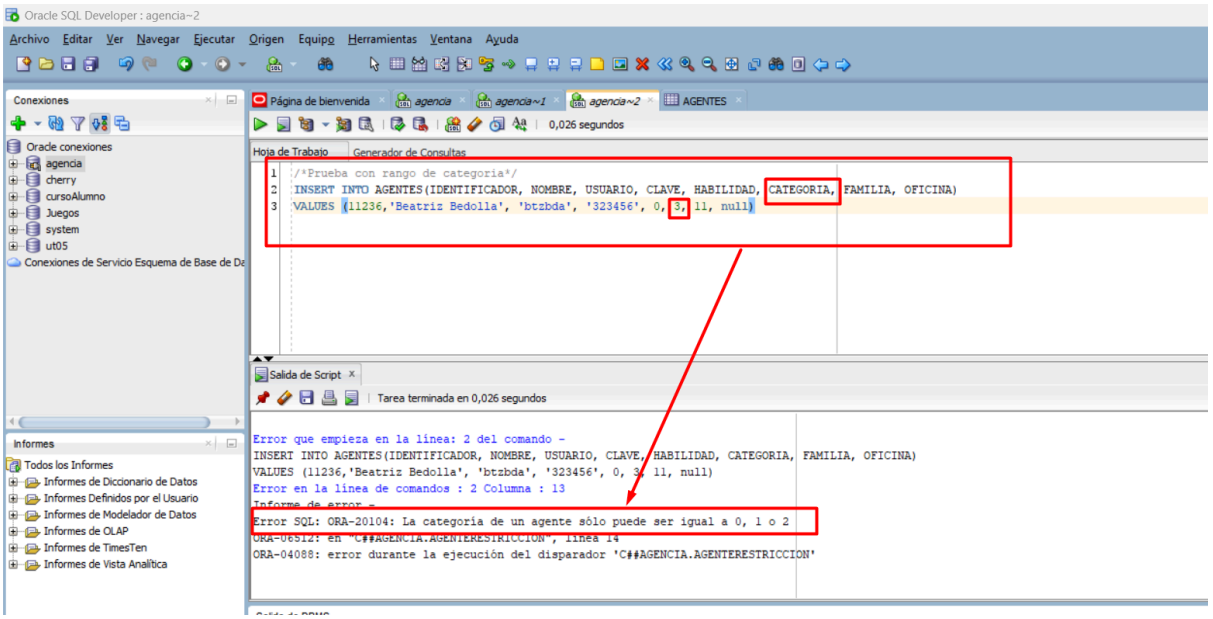
2.2 Probamos primera restricción de la clave de un agente: no puede ser inferior a 6 carácteres. El error se lanza . Cherry Reynoso Catalán.

2. **Habilidad de un agente comprendida entre 0 y 9 (ambos inclusive)**



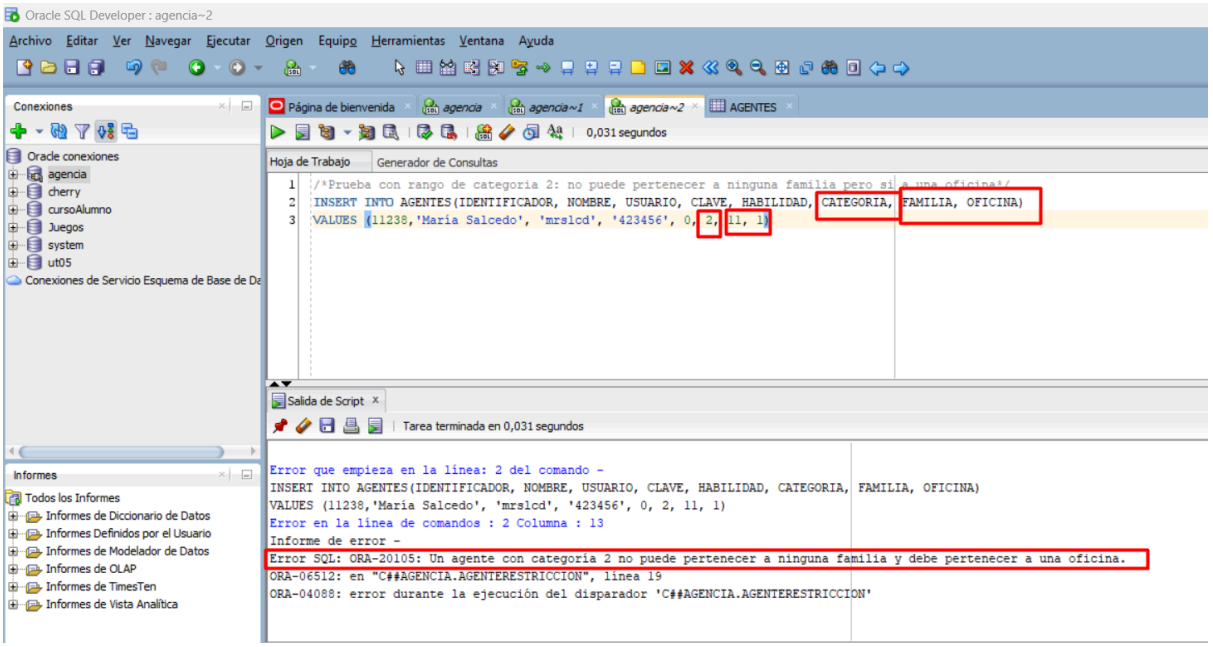
2.4 Probamos segunda restricción de la habilidad de un agente: debe ser entre 0 y 9. Un número negativo no es posible. El error se lanza. Cherry Reynoso Catalán.

3. **Rango de categoria: solo pueder ser 0,1 o 2**

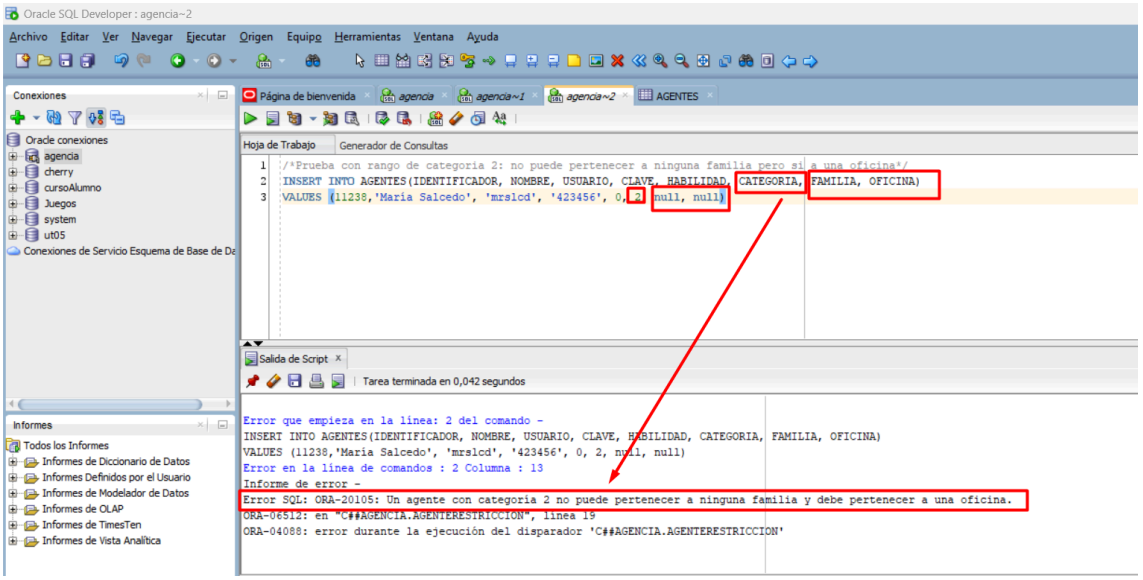


2.6 Probamos tercera restricción de la categoría de un agente:solo puede ser de 0 a 2.A partir de 3 el error se lanza. Cherry Reynoso Catalán.

4. **Restricción de pertenencia en categoria 2: no puede pertenecer a ninguna familia pero sí a una oficina.**

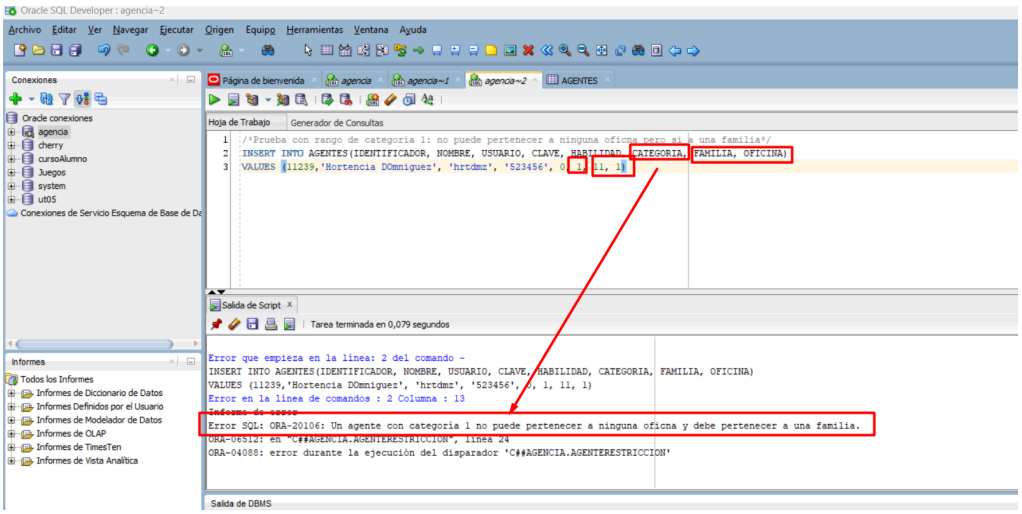


2.7 Probamos cuarta restricción de la categoría 2 un agente: no puede pertenecer a una familia. El error se lanza. Cherry Reynoso Catalán.

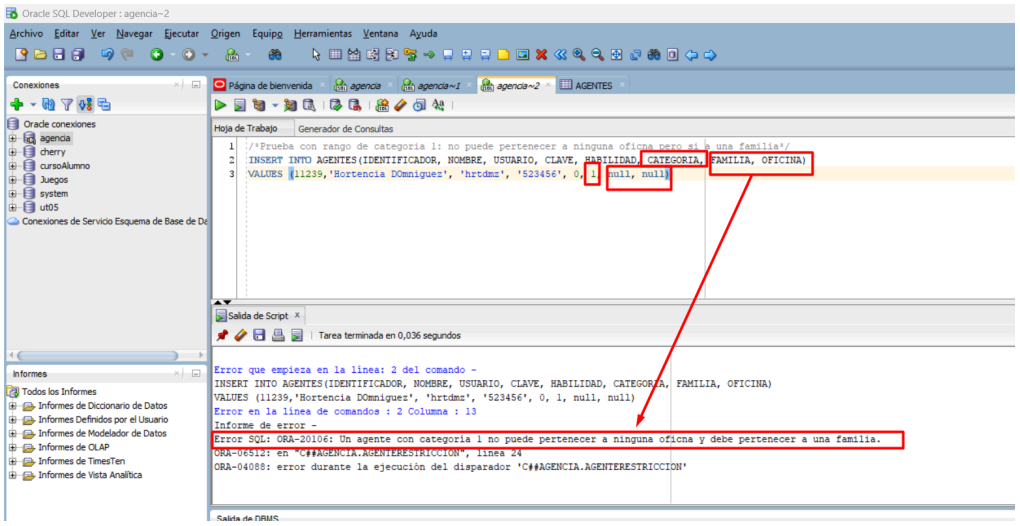


2.8 Probamos cuarta restricción de la categoría 2 un agente: no puede pertenecer a una familia pero debe pertenecer a una oficina. El error se lanza. Cherry Reynoso Catalán.

5. **Restricción de pertenencia en categoria 1: no puede pertenecer a ninguna oficina pero sí a una familia.**

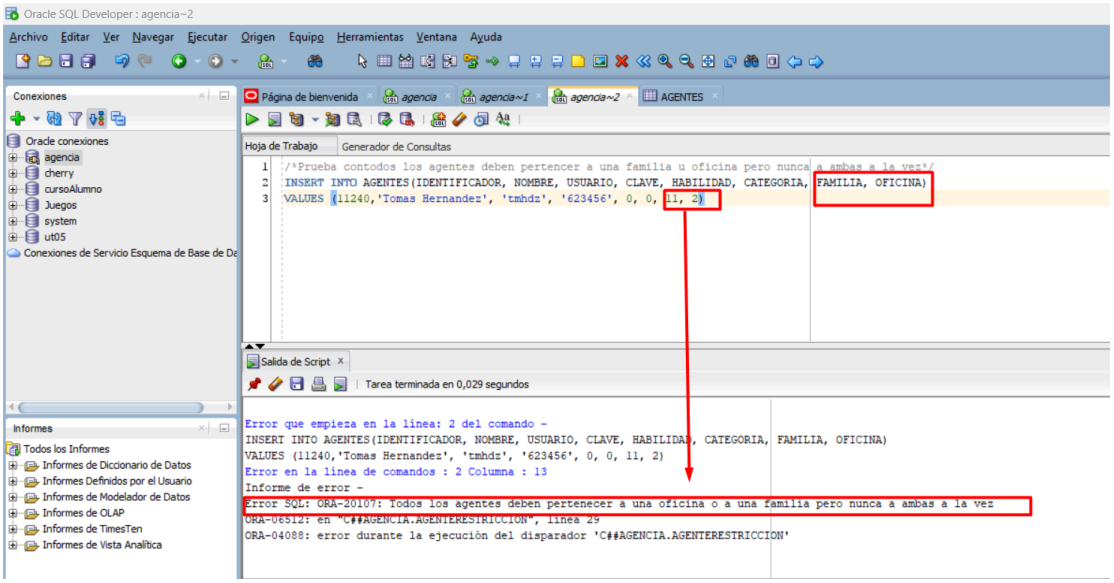


2.10 Probamos quinta restricción de la categoría 1 un agente: no puede pertenecer a una oficina. El error se lanza. Cherry Reynoso Catalán.

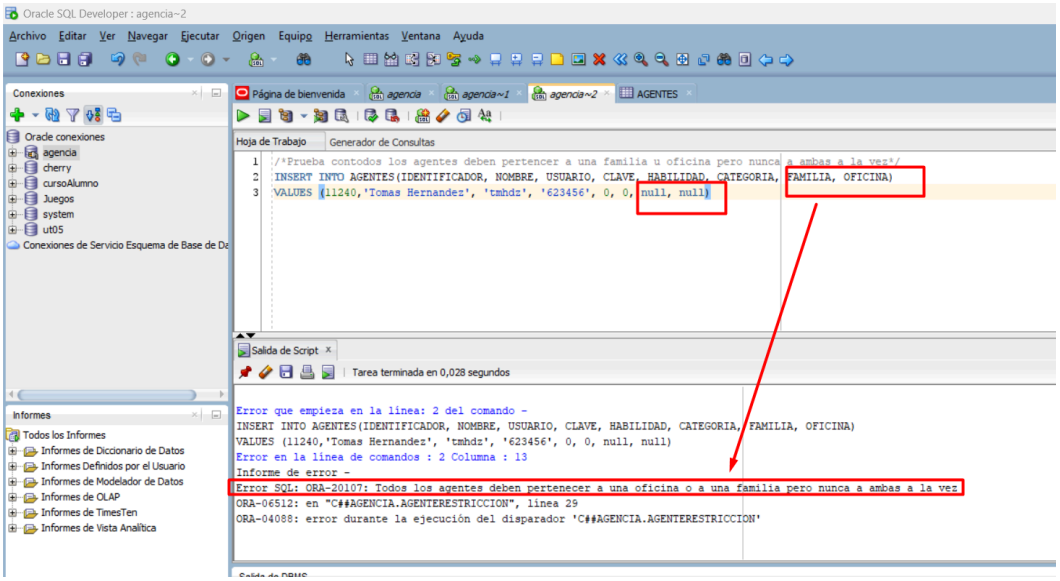


2.11 Probamos quinta restricción de la categoría 1 un agente: no puede pertenecer a una oficina pero debe pertenecer a una familia. El error se lanza. Cherry Reynoso Catalán.

6. Pertenencia a una oficina o una familia pero nunca a ambas a la vez.



2.13 Probamos sexta restricción: todos los agentes no pueden pertenecer a una familia y a una oficina a la vez. El error se lanza. Cherry Reynoso Catalán.



2.14 Probamos sexta restricción: todos los agentes deben pertenecer a una familia o a una oficina; no es posible valores null en ambas. El error se lanza. Cherry Reynoso Catalán.

***Algunas de las restricciones implementadas con el disparador se pueden incorporar a la definición del esquema de la tabla utilizando el Lenguaje de Definición de Datos (Check,Unique,..)*.Identifica cuáles son y con qué tipo de restricciones las implementarías.**

- TIPO RESTRICCION: **CHECK.**
- CONDICIÓN: `HABILIDAD BETWEEN 0 AND 9`

```
ALTER TABLE agentes
ADD CONSTRAINT chk_habilidad_rango CHECK (habilidad BETWEEN 0 AND
9);
```

- TIPO RESTRICCION: **CHECK.**
- CONDICIÓN: `CATEGORIA IN (0,1,2)`

```
ALTER TABLE agentes
ADD CONSTRAINT chk_categoria_rango CHECK (categoria IN (0, 1,
2));
```

CONCLUSION:
No es posible en la restricción de la clave en DDL porque en los datos anteriores YA HAY CLAVES CON MENOS DE 5 caracteres en los datos de la tabla.
Por otro lado las restricciones sobre oficinas y familias para todos los agentes, no puede modificarse ya que son FK y son condicionales (unos valores son NOT NULL y otros NULL). Esto nos demuestra la utilidad de los disparadores o triggers.

El esquema de la tabla usando DDL sería:

```
CREATE TABLE agentes (
  identificador NUMBER(6) NOT NULL PRIMARY KEY,
  nombre VARCHAR2(60) NOT NULL,
  usuario VARCHAR2(20) NOT NULL UNIQUE,
  clave VARCHAR2(20) NOT NULL CHECK (LENGTH(clave) >= 6),
  habilidad NUMBER(1) CHECK (habilidad BETWEEN 0 AND 9) NOT NULL,
  categoria NUMBER(1) CHECK (categoria IN (0, 1, 2)) NOT NULL,
  familia NUMBER(6) REFERENCES familias,
  oficina NUMBER(6) REFERENCES oficinas,
  CHECK ((categoria = 1 AND oficina IS NULL AND familia IS NOT NULL)
        OR (categoria = 2 AND oficina IS NOT NULL AND familia IS NULL)
        OR (categoria != 1 AND categoria != 2))
);
```