Ejercicio 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.

```
--Creo el procedimiento con dos parámetros del tipo del campo familias de la tabla agentes.

CREATE OR REPLACE PROCEDURE CambiarAgentesFamilia(
                        id_FamiliaOrigen agentes.familia%TYPE,
                        id_FamiliaDestino agentes.familia%TYPE)
AS
   /* Declaro un cursor variable que haga referencia a cada una de las
   filas de la tabla agentes y declaro una variable de tipo de ese cursor creado.*/

   TYPE cursor_familias IS REF CURSOR RETURN agentes%ROWTYPE;
   cFamilias cursor_familias;

   /* Declaro dos variables numéricas. Una me servirá para contar
   el número de agentes existentes a con el código de familia que se pase por parámetro.
   La otra me servirá para ver si existe la familia de destino a la que quiero modificar
   de los agentes. */

   v_NFamiliaOrigen number(8) :=0;
   v_NFamiliaDestino number(8) :=0;

BEGIN
   /*Recorro el cursor variable haciendo un select de los agentes existentes que tienen
   como familia el parámetro de la familia de origen y en cada vuelta del for sumo uno
   a la variable contadora.*/

   FOR cFamilias IN (SELECT * FROM agentes WHERE familia = id_FamiliaOrigen) LOOP
      v_NFamiliaOrigen := v_NFamiliaOrigen + 1;
   END LOOP;

   /*Recorro el cursor variable haciendo un select a las familias que coincidan con
   la variable de familia de destino y sumo uno a la variable contadora por cada vuelta del bucle.*/

   FOR cFamilias IN (SELECT * FROM familias WHERE identificador = id_FamiliaDestino) LOOP
      v_NFamiliaDestino := v_NFamiliaDestino + 1;
   END LOOP;


   --Primero analizo si la familia de destino existe y si no, lanzo una excepción.

   IF v_NFamiliaDestino = 0 THEN
```

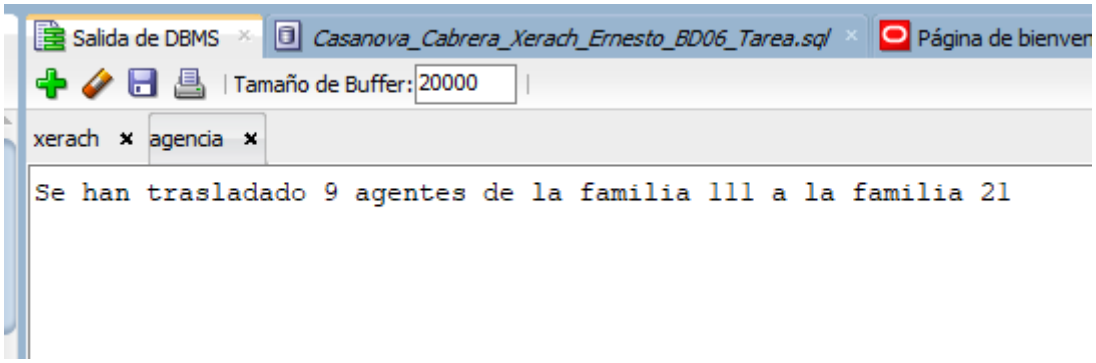
```
        RAISE_APPLICATION_ERROR(-20201, 'No hay códigos de familia que coincida con la familia
indicada.');

    /*Después analizo si existen agentes que pertenezcan a la familia que se indicó por parámetros
    y si no existe devuelvo una excepción.*/

    ELSIF v_NFamiliaOrigen = 0 THEN
        RAISE_APPLICATION_ERROR(-20202, 'No existen agentes con el código indicado.');

    /*Por último analizo si los parámetros de familia de origen y familia destino son iguales
    y si es así, devuelvo una excepción*/

    ELSIF id_FamiliaOrigen = id_FamiliaDestino THEN
        RAISE_APPLICATION_ERROR(-20203, 'La categoría de la familia de origen es idéntica a la de
destino. No se ha realizado ningún cambio.');

    /*Si ninguno de los casos anteriores se cumple, hacemos un update con la nueva familia para los
agentes seleccionados y mostramos mensaje por pantalla.*/

    ELSE
        UPDATE agentes SET familia = id_FamiliaDestino WHERE familia = id_FamiliaOrigen;
        DBMS_OUTPUT.PUT_LINE('Se han trasladado ' || v_NFamiliaOrigen || ' agentes de la familia '
                            || id_FamiliaOrigen || ' a la familia ' || id_FamiliaDestino);
    END IF;

END;
/
```

Para probar el procedimiento creo un bloque que llame al procedimiento pasándole dos parámetros que introduciremos por pantalla:

```
DECLARE

    p_id_FamiliaOrigen agentes.familia%TYPE := &familiaOrigen;
    p_id_FamiliaDestino agentes.familia%TYPE := &familiaDestino;

BEGIN

    CambiarAgentesFamilia(p_id_FamiliaOrigen, p_id_FamiliaDestino);

END;
/
```

Y fuerzo cada uno de los casos que se pueden producir:

Traslado de agentes de una familia a otra con éxito:



Error producido al no encontrar agentes con una familia que coincida con la pasada por parámetros.

```
Nuevo:DECLARE

    p_id_FamiliaOrigen agentes.familia%TYPE := 21;
    p_id_FamiliaDestino agentes.familia%TYPE := 1500;

BEGIN

    CambiarAgentesFamilia(p_id_FamiliaOrigen, p_id_FamiliaDestino);

END;

Error que empieza en la línea: 111 del comando :
DECLARE

    p_id_FamiliaOrigen agentes.familia%TYPE := &familiaOrigen;
    p_id_FamiliaDestino agentes.familia%TYPE := &familiaDestino;

BEGIN

    CambiarAgentesFamilia(p_id_FamiliaOrigen, p_id_FamiliaDestino);

END;
Informe de error -
ORA-20201: No hay códigos de familia que coincida con la familia indicada.
ORA-06512: en "C##AGENCIA.CAMBIARAGENTESFAMILIA", línea 36
ORA-06512: en línea 8
```

Error producido al no encontrar la familia de destino pasada por parámetros que coincida con alguna existente:

```
Nuevo:DECLARE

    p_id_FamiliaOrigen agentes.familia%TYPE := 500;
    p_id_FamiliaDestino agentes.familia%TYPE := 111;

BEGIN

    CambiarAgentesFamilia(p_id_FamiliaOrigen, p_id_FamiliaDestino);

END;

Error que empieza en la línea: 111 del comando :
DECLARE

    p_id_FamiliaOrigen agentes.familia%TYPE := &familiaOrigen;
    p_id_FamiliaDestino agentes.familia%TYPE := &familiaDestino;

BEGIN

    CambiarAgentesFamilia(p_id_FamiliaOrigen, p_id_FamiliaDestino);

END;
Informe de error -
ORA-20202: No existen agentes con el código indicado.
ORA-06512: en "C##AGENCIA.CAMBIARAGENTESFAMILIA", línea 42
ORA-06512: en línea 8
```

Error producido al introducir una familia de destino igual a la familia de origen:

```
Nuevo:DECLARE

    p_id_FamiliaOrigen agentes.familia%TYPE := 21;
    p_id_FamiliaDestino agentes.familia%TYPE := 21;

BEGIN

    CambiarAgentesFamilia(p_id_FamiliaOrigen, p_id_FamiliaDestino);

END;

Error que empieza en la línea: 111 del comando :
DECLARE

    p_id_FamiliaOrigen agentes.familia%TYPE := &familiaOrigen;
    p_id_FamiliaDestino agentes.familia%TYPE := &familiaDestino;

BEGIN

    CambiarAgentesFamilia(p_id_FamiliaOrigen, p_id_FamiliaDestino);

END;
Informe de error -
ORA-20203: La categoría de la familia de origen es idéntica a la de destino. No se ha realizado ningún cambio.
ORA-06512: en "C##AGENCIA.CAMBIARAGENTESFAMILIA", línea 48
ORA-06512: en línea 8
```

Actividad 2.

Queremos controlar algunas restricciones a la hora de trabajar con agentes:

La longitud de la clave de un agente no puede ser inferior a 6.

La habilidad de un agente debe estar comprendida entre 0 y 9 (ambos inclusive).

La categoría de un agente sólo puede ser igual a 0, 1 o 2.

Si un agente tiene categoría 2 no puede pertenecer a ninguna familia y debe pertenecer a una oficina.

Si un agente tiene categoría 1 no puede pertenecer a ninguna oficina y debe pertenecer a una familia.

Todos los agentes deben pertenecer a una oficina o a una familia pero nunca a ambas a la vez.

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.

```
/*Creamos el trigger con un before INSERT or UPDATE para
controlar las restricciones antes de insertar o modificar
cualquier fila de agentes.*/

CREATE OR REPLACE TRIGGER integridad_Agentes
BEFORE INSERT OR UPDATE ON agentes
FOR EACH ROW

BEGIN

    --Si el tamaño de la clave es menor que 6 caracteres lanzamos una excepción:

    IF LENGTH(:NEW.clave) < 6 THEN
        RAISE_APPLICATION_ERROR(-20201, 'La clave del agente no puede ser menor que 6.');

    --Si la habilidad no se encuentra entre 0 y 9 o si es nula, lanzamos una excepción:

    ELSIF (:NEW.habilidad < 0 OR :NEW.habilidad > 9) OR :NEW.habilidad IS NULL THEN
```

```
      RAISE_APPLICATION_ERROR(-20202, 'El valor de la habilidad del agente debe estar comprendido
entre 0 y 9.');

    --Si la categoría no se encuentra entre no se encuentra entre 0 y 2 o si es nula, lanzamos una
excepción:

   ELSIF :NEW.categoria < 0 OR :NEW.categoria  > 2 OR :NEW.categoria IS NULL THEN
      RAISE_APPLICATION_ERROR(-20203, 'El valor de la categoría del agente no puede ser nula y debe
debe ser 0, 1 o 2.');


   ELSIF :NEW.categoria = 2 THEN

    -- Si la categoría es 2 y la familia no es nula lanzamos una excepción.

      IF :NEW.familia IS NOT NULL THEN
         RAISE_APPLICATION_ERROR(-20204, 'Un empleado con categoría 2 no puede pertenecer a
ninguna familia');

   -- Si la categoría es 2 y la oficina es nula lanzamos una excepción.

      ELSIF :NEW.oficina IS NULL THEN
         RAISE_APPLICATION_ERROR(-20205, 'Un empleado con categoría 2 debe pertenecer a una
oficina');

      END IF;

   ELSIF :NEW.categoria = 1 THEN

      -- Si la categoría es 1 y la familia es nula lanzamos una excepción.

      IF :NEW.familia IS NULL THEN
         RAISE_APPLICATION_ERROR(-20206, 'Un empleado con categoría 1 debe pertenecer a  una
familia');

      -- Si la categoría es 1 y la oficina no es nula lanzamos una excepción.

      ELSIF :NEW.oficina IS NOT NULL THEN
         RAISE_APPLICATION_ERROR(-20207, 'Un empleado con categoría 1 no puede pertenecer a una
oficina');

      END IF;

   -- Si la oficina y la familia tienen contenido ambas lanzamos una excepción.

   ELSIF :NEW.familia IS NOT NULL AND :NEW.oficina IS NOT NULL THEN
         RAISE_APPLICATION_ERROR(-20208, 'Un empleado no puede pertenecer a una familia y a una
oficina a la vez.');

   -- Si la oficina y la familia son nulas ambas lanzamos una excepción.

   ELSIF :NEW.familia IS NULL AND :NEW.oficina IS NULL THEN
         RAISE_APPLICATION_ERROR(-20209, 'Un empleado debe pertenecer a una familia o a una
oficina.');

   END IF;
END;
```

Una vez compilado el disparador, para probar cada una de las restricciones podemos hacer los siguientes INSERT:

```
INSERT INTO AGENTES VALUES (1850, 'Xerach Casanova Cabrera', 'xcc', '12345', 0, 0, 1, null);
INSERT INTO AGENTES VALUES (1850, 'Xerach Casanova Cabrera', 'xcc', '123456', -1, 0, 1, null);
INSERT INTO AGENTES VALUES (1850, 'Xerach Casanova Cabrera', 'xcc', '123456', -9, 0, 1, null);
INSERT INTO AGENTES VALUES (1850, 'Xerach Casanova Cabrera', 'xcc', '123456', 0, -1, 1, null);
INSERT INTO AGENTES VALUES (1850, 'Xerach Casanova Cabrera', 'xcc', '123456', 0, 3, 1, null);
INSERT INTO AGENTES VALUES (1850, 'Xerach Casanova Cabrera', 'xcc', '123456', 0, 2, 1, null);
INSERT INTO AGENTES VALUES (1850, 'Xerach Casanova Cabrera', 'xcc', '123456', 0, 1, null, 1);
INSERT INTO AGENTES VALUES (1850, 'Xerach Casanova Cabrera', 'xcc', '123456', 0, 1, null, null);
INSERT INTO AGENTES VALUES (1850, 'Xerach Casanova Cabrera', 'xcc', '123456', 0, 1, 1, 1);
```

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.

Las restricciones que podemos utilizar en el DDL podrían ser:

CHECK (clave.length >= 6) → La clave debe ser mayor o igual que 6 caracteres.
CHECK (habilidad BETWEEN 0 AND 9) → la habilidad debe estar comprendida entre 0 y 9
CHECK (categoría BETWEEN 0 AND 2 → la categoría debe estar comprendida entre 0 y 2.