CENTRE ESPECÍFIC
D´EDUCACIÓ A DISTÀNCIA DE
LA COMUNITAT VALENCIANA

TAREA EVALUABLE 3 SOLUCIÓN PROPUESTA

Bases de Datos
CFGS DAW

Pau Miñana Climent
2020/2021

Gestión de Usuarios (2,5 puntos)

1. Añadir el campo usuario a la tabla usuarios, de tipo VARCHAR(6), con la restricción de que el nombre de usuario no se puede repetir

   ```
   ALTER TABLE evaluable3.usuarios ADD (usuario varchar(6) DEFAULT NULL UNIQUE);
   ```

2. Rellenar mediante PL/SQL todos los usuarios con las 3 primeras letras del nombre y las 3 primeras letras del apellido1 (es una mala decisión de diseño pues es fácil que se repitan, pero en este caso nos vale). Para ello podéis usar la función SUBSTR(cadena,num_inicio,num_fin) que extrae "substrings" de una cadena.

   ```
   BEGIN
       FOR usu IN (SELECT * FROM evaluable3.usuarios) LOOP
           update evaluable3.usuarios
           SET usuario=SUBSTR(usu.nombre,1,3)||SUBSTR(usu.apellido1,1,3)
           WHERE dni=usu.dni;
       END LOOP;
   END;
   /
   ```

3. Crear los roles profesor y alumno y un usuario alumno de ejemplo al que asignaremos sólo el permiso para crear una sesión y el rol correspondiente.

   ```
   CREATE ROLE ALUMNO;
   CREATE ROLE PROFESOR;
   GRANT CREATE SESSION TO ALUMNO,PROFESOR;
   CREATE USER FERFER IDENTIFIED BY fernando
   DEFAULT TABLESPACE evaluable3;
   GRANT ALUMNO TO FERFER;
   ```

4. Dar al rol alumno permiso para ejecutar las funciones procedimientos que se correspondan.

   ```
   GRANT EXECUTE ON evaluable.muestra_modulos TO ALUMNO;
   GRANT EXECUTE ON evaluable.muestra_ti TO ALUMNO;
   GRANT EXECUTE ON evaluable.muestra_reservas TO ALUMNO;
   GRANT EXECUTE ON evaluable.anula_tutoria TO ALUMNO;
   GRANT EXECUTE ON evaluable.muestra_ti_libres TO ALUMNO;
   ```

Gestión de la BD (3 puntos)

1. *evaluable3.dia_sem*: Pasa el día de la semana de un número al nombre.

```
CREATE OR REPLACE FUNCTION evaluable3.dia_sem(dia IN INT)
RETURN VARCHAR
IS
BEGIN
        CASE dia
                WHEN 2 THEN
                        RETURN 'LUNES';
                WHEN 3 THEN
                        RETURN 'MARTES';
                WHEN 4 THEN
                        RETURN 'MIERCOLES';
                WHEN 5 THEN
                        RETURN 'JUEVES';
                WHEN 6 THEN
                        RETURN 'VIERNES';
                ELSE
                        RETURN 'ERROR';
        END CASE;
END;
/
```

   Aquí se podría lanzar una excepción en el ELSE puesto que el día en la tabla TI, que es donde se usará tiene que contener por definición uno de los números anteriores. Pero no lo considero necesario, sobre todo considerando que es una función dirigida a usuarios sin permisos y que no es un fallo crítico.

2. Un trigger para la actualización en cascada del departamento en profesores.

```
CREATE OR REPLACE TRIGGER dep_cascada
BEFORE UPDATE
ON evaluable3.departamentos
FOR EACH ROW when (old.departamento<>new.departamento)
BEGIN
        UPDATE profesores
        SET departamento=:new.departamento
        WHERE departamento=:old.departamento;
        DBMS_OUTPUT.PUT_LINE('Dep. actualizado en profesores');
END;
/
```

3. Un trigger para la actualización en cascada del cod_mod en imparten y en matriculados cuando se cambie en modulos.

```
CREATE OR REPLACE TRIGGER mod_cascada
BEFORE UPDATE
ON evaluable3.modulos
FOR EACH ROW when (old.cod_mod<>new.cod_mod)
BEGIN
        UPDATE matriculados
        SET cod_mod=:new.cod_mod
        WHERE cod_mod=:old.cod_mod;
        UPDATE imparten
        SET cod_mod=:new.cod_mod
        WHERE cod_mod=:old.cod_mod;
        DBMS_OUTPUT.PUT_LINE('Codigo actualizado en imparten/matriculados');
END;
/
```

4. Un trigger para evitar que un profesor pueda estar matriculado como alumno en un módulo que lo imparta alguien de su mismo departamento (tanto al añadir datos como al cambiarlos).

Primera opción, borrar las tuplas erróneas después de la operación insert/update:

```
CREATE OR REPLACE TRIGGER mat_prof
AFTER INSERT OR UPDATE
ON evaluable3.matriculados
BEGIN
  FOR borrar in (SELECT UNIQUE p.dni,m.cod_mod
                FROM profesores p, profesores p2, alumnos a,matriculados m, imparten i
                WHERE p.dni=a.dni AND p.dni=m.dni_alumno
                AND p2.departamento=p.departamento AND p2.dni=i.dni_profesor
                AND m.cod_mod=i.cod_mod)
  LOOP
          DELETE FROM evaluable3.matriculados
          WHERE borrar.dni=dni_alumno AND borrar.cod_mod=cod_mod;
  END LOOP;
END;
/
```

Pro: Es más rápido al cambiar los datos una vez acabada la operación y no comprobar fila a fila. Permite un insert/update múltiple y sólo borra los erróneos, los demás sí se modifican. Corrige errores anteriores puesto que simplemente borra los que no cumplen las condiciones, es decir que el profesor_alumno con un módulo que no puede tener que ya existía en la BBDD se borrará tras cualquier insert/update.

Contra: En un update que modifique el cod_mod de un alumno_profesor, no lo devuelve a su estado anterior sino que lo borra, así que se pierde el módulo antiguo y eso puede ser bueno o malo. Técnicamente, así los datos erróneos llegan a estar en la BD unos ms hasta que se ejecuta el trigger.

Otra opción, evitar con una excepción la operación errónea antes de que se produzca:

```
CREATE OR REPLACE TRIGGER mat_prof_alt
BEFORE INSERT OR UPDATE
ON evaluable3.matriculados
FOR EACH ROW
DECLARE
        dni_prof profesores.dni%TYPE;  -- para comprobar si el matriculado es profesor
        dep profesores.departamento%TYPE; -- departamento del prof_alumno matriculado
BEGIN
        -- si el matriculado no es profesor no devuelve nada y salta una excepcion
        -- NO_DATA_FOUND
        SELECT dni INTO dni_prof FROM profesores WHERE :NEW.dni_alumno=dni;
        -- si llega aquí es que no hay excepción y el dni era de un profesor
        SELECT departamento INTO dep FROM profesores WHERE :NEW.dni_alumno=dni;
        -- se hace un bucle con todos los módulos que imparten en el departamento y se
        -- comprueba en cada iteración que el módulo a insertar/actualizar no es el mismo.
        FOR modulo IN (SELECT UNIQUE cod_mod FROM profesores p, imparten i
                        WHERE p.dni=i.dni_profesor AND p.departamento=dep)
        LOOP
                -- si el módulo es el mismo que alguno en la lista se rompe el trigger y con
                -- ello el insert/update entero, ya que no tratamos esta excepción
                IF (:NEW.cod_mod=modulo.cod_mod) THEN
                        Raise_Application_Error (-20000, modulo.cod_mod||
                                                ' impartido por el mismo departamento.');
                END IF;
        END LOOP;
        EXCEPTION
                -- tratamos el no_data found para que siga comprobando las demás túplas
                -- de la operación y que el proceso continue, llegar aquí sólo significa que se
                -- están cambiando datos de un alumno que no es profesor, así que el
                -- proceso debe continuar y el trigger no hacer nada de momento.
                WHEN NO_DATA_FOUND THEN
                        dni_prof:=NULL; -- por dar algún valor, esto es irrelevante.
END;
/
```

Pro: Los datos erróneos nunca llegan a la BD.
Contra: Se fuerza una excepción sin tratamiento para romper una operación, lo que no es muy apropiado y puede romper procesos/scripts que realicen varias operaciones.
Se aborta la operación insert/update entera, esto puede ser bueno o malo pues por un lado los datos que sí pueden cambiar no lo hacen y por el otro, es probable que, puesto que la operación incluye cambios que no cumplen las condiciones, la operación insert/update esté mal planteada y que no se cambie ningún dato por la misma sea más apropiado ya que los otros aunque cumplan las condiciones también podrían ser un error.

Procedimientos de Alumnos (4,5 puntos)

El nombre del propietario evaluable3 no es necesario siempre que quien cree el procedimiento sea ese usuario. Tampoco dentro del mismo para las tablas o tipos, pero se ha dejado así en ocasiones para remarcar como trabajar con objetos de otros usuarios/bd.
Donde sí resulta imprescindible *evaluable3.nombre_procedimiento* es al lanzar el procedimiento desde otros usuarios, como el alumno que hemos creado.

1. *evaluable3.muestra_modulos*: Muestra los módulos del alumno que lo ejecuta

```
CREATE OR REPLACE PROCEDURE evaluable3.muestra_modulos
IS
id evaluable3.usuarios.dni%TYPE;
BEGIN
        SELECT dni INTO id FROM usuarios WHERE usuario=user;
        FOR matr IN (SELECT * FROM matriculados WHERE dni_alumno=id)
        LOOP
                DBMS_OUTPUT.PUT_LINE(matr.cod_mod);
        END LOOP;
END;
/
```

Para este procedimiento y los siguientes: se puede obviar la variable id y usar ese primer SELECT como subconsulta en el SELECT del bucle, donde pone “=id”.
Técnicamente al hacer un insert INTO deberíamos tratar la excepción NO_DATA_FOUND que se puede producir si el user no está en la lista de usuarios, pero vamos a suponer que esto no puede pasar ya que se comprobaría al iniciar sesión en la supuesta aplicación. Aún así, esto es sólo por no alargar la solución, es una mala praxis; estas comprobaciones de seguridad conviene hacerse SIEMPRE en casos reales porque pueden haber errores inesperados de cualquier tipo, por ejemplo, corrupción de datos en el proceso de comunicación entre la app y la BD y además la aplicación no recibiría respuesta de la BD en caso de error.

2. *evaluable3.muestra_ti*: Se supone que al pinchar en los módulos devueltos por la función anterior se puede ver el horario de ti del profesor. Por tanto esta función muestra el horario SEMANAL del profesor(o profesores) del módulo que se le pasa como parámetro.

```
CREATE OR REPLACE PROCEDURE evaluable3.muestra_ti(modulo IN varchar)
IS
BEGIN
        FOR tutoria IN ( SELECT dia_sem(ti.dia) as dia,
                        TO_CHAR(hora, 'HH24:MI') as hora,  ti.dni_profesor
                        FROM ti,imparten i
                        WHERE ti.dni_profesor=i.dni_profesor AND cod_mod=modulo)
        LOOP
                DBMS_OUTPUT.PUT_LINE(tutoria.dia||'  '||tutoria.hora||
                '  '||tutoria.dni_profesor);
        END LOOP;
END;
/
```

3. *evaluable3.muestra_reservas*: Muestra las reservas de ti que ha hecho el usuario. Pueden ser todas o sólo las que aún no se han realizado.
   En este caso se muestran sólo las que tienen una fecha posterior al día actual. Aunque se filtren posteriormente con un IF también se podría haber añadido la condición directamente en el WHERE de la consulta anterior.

```
CREATE OR REPLACE PROCEDURE evaluable3.muestra_reservas
IS
id evaluable3.usuarios.dni%TYPE;
BEGIN
        SELECT dni INTO id from evaluable3.usuarios WHERE usuario=user;
        FOR reserva IN (SELECT fecha_ti,
                        TO_CHAR(hora, 'HH24:MI') as hora,dni_profesor
                         FROM reserva_ti rti NATURAL JOIN ti WHERE dni_alumno=id
                          ORDER BY fecha_ti)
        LOOP
                IF (reserva.fecha_ti>=SYSDATE()) THEN
                        DBMS_OUTPUT.PUT_LINE(reserva.fecha_ti||'  '||reserva.hora
                        ||'  '||reserva.dni_profesor);
                END IF;
        END LOOP;
END;
/
```

4. *evaluable3.anula_tutoría*: Borra una tutoría ya existente del usuario actual. Obviamente necesita la fecha y id_ti como parámetros.

```
CREATE OR REPLACE PROCEDURE evaluable3.anula_tutoria(id_reserva IN VARCHAR,
                                                     fecha IN DATE)
IS
        id evaluable3.usuarios.dni%TYPE;
        reserva evaluable3.reserva_ti%ROWTYPE;
BEGIN
        SELECT dni INTO id FROM evaluable3.usuarios WHERE usuario=user;
        SELECT *  INTO reserva FROM reserva_ti
        WHERE dni_alumno=id AND fecha_ti=fecha  AND id_ti=id_reserva;
        DELETE FROM reserva_ti
        WHERE fecha_ti=reserva.fecha_ti AND id_ti=reserva.id_ti;
        EXCEPTION
                WHEN NO_DATA_FOUND THEN
                DBMS_OUTPUT.PUT_LINE('No se ha anulado ninguna reserva');
END;
/
```

En esta solución se realizan algunas comprobaciones extra de seguridad. La parte del dni del usuario puede obviarse, pero esto implicaría que un usuario podría borrar las reservas de otro si hay un error en los parámetros de entrada. Además se considera que dichos parámetros podrían no coincidir con ninguna reserva del alumno. Estas 2 posibilidades harían que las consultas no devolviesen nada y salta la excepción “no data found”. Por esta vez se decide tratarla y simplemente avisar de que no se ha anulado la reserva correspondiente. Considerad que esto resulta útil pues se puede enviar una respuesta adecuada a la aplicación. Sin tratar excepciones, en caso de errores la aplicación puede quedar bloqueada.

5. *evaluable3.muestra_ti_libres:* Muestra las ti libres en las próximas 2 semanas del módulo que se le pasa como parámetro.

Antes que nada creamos la tabla ti_modulo donde guardaremos los datos auxiliares. Se pueden crear variables tipo tabla, en realidad, pero no se han visto en el curso así que no se usarán.

```
CREATE TABLE ti_modulo(
        cod VARCHAR(4),
        fecha DATE,
        dia INTEGER,
        hora VARCHAR(5),
        dni_profesor VARCHAR(9)
);
```

Se procede ahora con el procedimiento para guardar y mostrar las reservas:

```
CREATE OR REPLACE PROCEDURE evaluable3.muestra_ti_libres(modulo in varchar)
IS
BEGIN
    -- con este bucle se insertan en la tabla las tutorias de los proximos 14 días, uno por
    -- cada iteracion del bucle
    FOR I IN 0 .. 13
    LOOP
        -- se guardan los datos de las ti correspondientes al modulo con fecha del día
        -- correspondiente, TO_DATE(SYSDATE()+I), asi cada iteracion para a ser el día
        -- siguiente. Para asegurar que las tutorías son solo las que se corresponden a
        -- ese día , la condicion TO_CHAR(SYSDATE()+I, 'D')=dia) transforma a numero
        -- el día y compara con el numero de dia en la tabla TI.
        INSERT INTO ti_modulo (SELECT id_ti as cod,TO_DATE(SYSDATE()+I) as fecha,
                                dia,TO_CHAR(hora, 'HH24:MI') as hora, ti.dni_profesor
                                FROM ti,imparten i
                                WHERE ti.dni_profesor=i.dni_profesor
                                AND cod_mod=modulo
                                AND TO_CHAR(SYSDATE()+I, 'D')=dia);
    END LOOP;

    -- se borran de la tabla las tuplas que coinciden con reservas de la tabla reservas_ti
    DELETE FROM ti_modulo
    WHERE (fecha,cod) IN (SELECT fecha_ti,id_ti FROM ti_modulo,reserva_ti
                        WHERE fecha_ti=fecha AND id_ti=cod);

    -- se muestra por pantalla el resultado final y se limpia la tabla auxiliar para que no
    -- contenga datos y se pueda usar de nuevo, en caso contrario las reservas se
    -- conservan en la tabla y cada vez que se llama al procedimiento se van sumando
    -- otras, en vez de contener solo los valores deseados.
    FOR tutoria IN (SELECT * from ti_modulo)
    LOOP
        DBMS_OUTPUT.PUT_LINE(tutoria.fecha|| ' '||tutoria.hora|| ' '
                            ||tutoria.dni_profesor);
    END LOOP;
    DELETE FROM ti_modulo;
END;
/
```