UNIVERSIDAD
NACIONAL
DE COLOMBIA

Bases de datos II

Primer trabajo PL/SQL

Diego Valentín Osorio Marín
Jan Michael Muñóz Botero
Santiago Castro Tabares

Septiembre 2022

1. Configuración inicial del código

Para poder ejecutar las pruebas al código, por favor ejecutar los archivos en el siguiente orden:

a) *tablesCreation.sql:* Para crear las tablas del enunciado.
b) *packageUtil.sql:* Definición de tipos utilizados en todo el programa.
c) *triggerCoope.sql:* Triggers asociados a la tabla cooperativa.
d) *triggerCoopeSocio.sql:* Triggers asociados a la tabla coopexsocio.
e) *triggerSocio.sql:* Trigger asociados a la tabla socio.
f) *packagePrograms:* De aquí se pueden llamar los procedimientos para los programas uno y dos.

Una vez hecho esto, proceda a ejecutar sus casos de prueba deseados.

2. Trigger de inserción tablas cooperativa, socio y coopexsocio

Se utilizó un trigger de tipo fila y momento before, donde simplemente se controla que:

1. Si el valor nuevo del acumulado a ingresar es NULL o es diferente de cero, entonces se cambie este valor a cero.

3. Trigger de actualización sobre la tabla cooperativa

Se utilizó un trigger de tipo fila y momento before, donde:

1. Se calcula la diferencia entre el valor nuevo de acumulado y el valor antiguo obteniendo así el incremento en la cooperativa.
2. En los casos en que esta diferencia sea cero o negativa, se definen excepciones para controlar esto.
3. Por otro lado, cuando el incremento es mayor a cero, obtenemos los socios que pertenecen a la cooperativa a actualizar, guardando estos registros en un arreglo mediante una sentencia BULK COLLECT de la siguiente manera:

```
SELECT SOCIO, COOPE, SC_ACUMULADO
BULK COLLECT INTO arrayCoopexsocio
FROM COOPEXSOCIO
WHERE COOPE = coopeCodigo;
```

4. Luego obtenemos el incremento unitario, es decir cuanto le toca a cada socio en su acumulado, dividiendo el incremento total sobre la cantidad de socios, de la siguiente forma:

```
incrementoUnitario := incrementoTotal/arrayCoopexsocio.COUNT;
```

5. Por último, actualizamos las tablas socio y coopexsocio con los incrementos unitarios que le corresponden a cada uno, utilizando sentencias forall, de la siguiente forma:

```
forall i in 1..arrayCoopexsocio.COUNT
  UPDATE SOCIO
  SET S_ACUMULADO = S_ACUMULADO + incrementoUnitario
  WHERE IDSOCIO = arrayCoopexsocio(i).SOCIO;
```

```
forall i in 1..arrayCoopexsocio.COUNT
  UPDATE COOPEXSOCIO
  SET SC_ACUMULADO = SC_ACUMULADO + incrementoUnitario
  WHERE SOCIO = arrayCoopexsocio(i).SOCIO
  AND COOPE = coopeCodigo;
```

4. Trigger de borrado sobre la tabla cooperativa

Se utilizó un trigger de tipo fila y momento before, donde:

1. Primero obtenemos los socios que pertenecen a la cooperativa a ser borrada, declarando un arreglo y guardando estas filas en el mediante la sentencia BULK COLLECT de la siguiente manera:

```
SELECT SOCIO, COOPE, SC_ACUMULADO
BULK COLLECT INTO arrayCoopexsocio
FROM COOPEXSOCIO WHERE COOPE = coopeCodigo;
```

2. la sentencia forall, decrementamos en sc_acumulado el s_acumualdo de cada uno de los socios afectados por el borrado, de la siguente manera:

```
forall i in 1..arrayCoopexsocio.COUNT
  UPDATE SOCIO
  SET S_ACUMULADO = S_ACUMULADO - arrayCoopexsocio(i).SC_ACUMULADO
  WHERE IDSOCIO = arrayCoopexsocio(i).SOCIO;
```

3. Finalmente, borramos los socios de la tabla coopexsocio

```
DELETE FROM COOPEXSOCIO WHERE COOPE = coopeCodigo;
```

5. Trigger de borrado sobre la tabla socio

Se utilizó un trigger de tipo fila y momento before, donde:

1. Simplemente borramos de la tabla coopexsocio los registros donde el id del socio a borrar coincidan, de la siguiente forma:

```
DELETE FROM COOPEXSOCIO WHERE SOCIO = :old.IDSOCIO;
```

6. Primer programa

Se crea un procedimiento llamado uno que recibe el código de una cooperativa, dentro de este procedimiento se crea un cursor llamado *header* para guardar la query SQL (en esta hacemos un JOIN donde la columna coope de la tabla COOPEXSOCIO sea igual al código ingresado al procedimiento y de este JOIN guardamos el nombre de la columna SOCIO y el sc_acumulado de COOPEXSOCIO).

```
cursor header IS
  SELECT NOMBRE, C2.SC_ACUMULADO
  FROM SOCIO S JOIN COOPEXSOCIO C2
  ON S.IDSOCIO = C2.SOCIO
  WHERE c2.COOPE = codigoCoope;+
```

Luego creamos un sub bloque en donde realizamos una query SQL (en esta hacemos un INNER JOIN donde la columna codigo de COOPERATIVA sea igual a la columna coope de la tabla COOPEXSOCIO y coope sea igual al codigo ingresado al procedimiento y de este INNER JOIN guardamos el nombre, el c_acumulado de la tabla COOPERATIVA y hacemos un count(*) (esto con la intención de contar el número de socios en esta cooperativa), luego controlamos la excepción en el caso de que no encuentre socios para la cooperativa ingresada.

```
begin
  SELECT DISTINCT nombre, C_ACUMULADO, COUNT(*)
  INTO nameCoope, cAcumulado, cantSocios
  FROM COOPERATIVA INNER JOIN COOPEXSOCIO C2
  ON COOPERATIVA.CODIGO = C2.COOPE AND c2.COOPE = codigoCoope
  GROUP BY nombre, C_ACUMULADO;
exception
when NO_DATA_FOUND then
  SELECT NOMBRE, C_ACUMULADO
  INTO nameCoope, cAcumulado
  FROM COOPERATIVA
  WHERE CODIGO = codigoCoope;
  cantSocios := 0;
end;
```

Luego procedemos a imprimir todo en el orden indicado en el trabajo, imprimimos el nombre de la cooperativa, el c_acumulado, la cantidad de socios y luego para imprimir los socios de la cooperativa usamos el cursor en donde cada fila tenemos el nombre del socio y el sc_acumulado y en otra variable vamos acumulando el sc_acumulado para imprimirlo al final.

```
for i in header loop
  DBMS_OUTPUT.PUT_LINE(
    cont||'. (Nombre: '||i.NOMBRE||', Valorsc: '||i.SC_ACUMULADO||')'
  );
  totalAcumulado := totalAcumulado + i.SC_ACUMULADO;
  cont := cont + 1;
end loop;
```

Por último controlamos la excepción de que el codigo ingresado por el usuario no corresponda a ninguna cooperativa existente y el others al final.

```
exception
  when NO_DATA_FOUND then
    DBMS_OUTPUT.PUT_LINE(SQLCODE||' La Cooperativa ingresada no existe');
  when others then
    DBMS_OUTPUT.PUT_LINE(SQLCODE || ' ' || SQLERRM);
end;
```

7. Segundo programa

En el paquete program se encuentra el procedimiento dos, que tiene como entrada el parámetro del código del socio (*codigoSocio*) del cual se va a mostrar la información requerida. Para obtener todas las cooperativas a las que pertenece el socio se crea un cursor (*header*) con la siguiente query SQL:

```
cursor header is
  SELECT C4.NOMBRE, C3.SC_ACUMULADO
  FROM COOPERATIVA C4 INNER JOIN COOPEXSOCIO C3
  ON C4.CODIGO = C3.COOPE
  WHERE c3.SOCIO = codigoSocio;
```

En la cual se obtiene el nombre y el acumulado de la cooperativa en la cual participa el socio con el código. Para obtener todas las cooperativas en las que no participa el socio, se crea un cursor (*footer*) con la siguiente query SQL:

```
cursor footer is
  SELECT NOMBRE
  FROM cooperativa LEFT JOIN COOPEXSOCIO C2
  ON COOPERATIVA.CODIGO = C2.COOPE AND C2.SOCIO = codigoSocio
  WHERE SOCIO IS NULL;
```

En el cual se obtiene el nombre de todas las cooperativas que tienen como socio null con la ayuda de un left join Una vez definidos los dos cursores para obtener las cooperativas a las que pertenece y a las que no, se obtiene mediante un query SQL para obtener el nombre, el acumulado y la cantidad de cooperativas en las que participa, dicha query se encapsula en un sub bloque para controlar el error: NO_DATA_FOUND y poder manejar el caso en el que el socio no participa en ninguna cooperativa

La impresión se lleva acabo abriendo y manipulando los cursores con el uso del loop for que se encuentran en el bloque más alto para poder manejar el error NO_DATA_FOUND pero en este caso este se dispara solo cuando el código del socio ingresado no se encuentra en la base de datos (Similar a como se manejó en el primer programa).