



Bases de datos II

Primer trabajo PL/SQL

Diego Valentín Osorio Marín
Jan Michael Muñoz 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:

- 1.1 *tablesCreation.sql*: Este archivo crea las tablas que se utilizarán en el proyecto.
- 1.2 *packageUtil.sql*: Este archivo crea el paquete donde se definen los tipos utilizados en todo el programa.
- 1.3 *triggerCoope.sql*: Este archivo crea los triggers asociados a la tabla cooperativa.
- 1.4 *triggerCoopeSocio.sql*: Este archivo crea los triggers asociados a la tabla coopexsocio.
- 1.5 *triggerSocio.sql*: Este archivo crea los triggers asociados a la tabla socio.
- 1.6 *packagePrograms*: Este archivo crea el paquete donde se definen los procedimientos utilizados para el programa uno y dos.

2. Trigger de inserción tablas cooperativa, socio y coopexsocio

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

- 2.1 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:

- 3.1 Se calcula la diferencia entre el valor nuevo de acumulado y el valor antiguo obteniendo así el incremento en la cooperativa.
- 3.2 En los casos en que esta diferencia sea cero o negativa, se definen excepciones para controlar esto.
- 3.3 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:

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

- 3.4 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:

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

- 3.5 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:

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

1 forall i in 1..arrayCoopexsocio.COUNT
2   UPDATE COOPEXSOCIO
3   SET SC_ACUMULADO = SC_ACUMULADO + incrementoUnitario
4   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:

- 4.1 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:

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

- 4.2 la sentencia forall, decrementamos en sc_acumulado el s_acumulado de cada uno de los socios afectados por el borrado, de la siguiente manera:

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

- 4.3 Finalmente, borramos los socios de la tabla coopexsocio

```
1 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:

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

```
1 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).

```
1 cursor header IS
2   SELECT S.NOMBRE, C2.SC_ACUMULADO
3   FROM SOCIO S JOIN COOPEXSOCIO C2
4   ON S.IDSOCIO = C2.SOCIO
5   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.

```
1 begin
2   SELECT DISTINCT C1.NOMBRE, C1.C_ACUMULADO, COUNT(*)
3   INTO nameCoope, cAcumulado, cantSocios
4   FROM COOPERATIVA C1 INNER JOIN COOPEXSOCIO C2
5   ON C1.CODIGO = C2.COOPE AND C2.COOPE = codigoCoope
6   GROUP BY NOMBRE, C_ACUMULADO;
7 exception
8 when NO_DATA_FOUND then
9   SELECT NOMBRE, C_ACUMULADO
10  INTO nameCoope, cAcumulado
11  FROM COOPERATIVA
12  WHERE CODIGO = codigoCoope;
13  cantSocios := 0;
14 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.

```

1 for i in header loop
2   DBMS_OUTPUT.PUT_LINE(
3     cont||'. (Nombre: '||i.NOMBRE||', Valorsc: '||i.SC_ACUMULADO||')'
4   );
5   totalAcumulado := totalAcumulado + i.SC_ACUMULADO;
6   cont := cont + 1;
7 end loop;

```

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

```

1 exception
2   when NO_DATA_FOUND then
3     DBMS_OUTPUT.PUT_LINE(SQLCODE||' La Cooperativa ingresada no existe');
4   when others then
5     DBMS_OUTPUT.PUT_LINE(SQLCODE || ' ' || SQLERRM);
6 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:

```

1 cursor header is
2   SELECT C4.NOMBRE, C3.SC_ACUMULADO
3   FROM COOPERATIVA C4 INNER JOIN COOPEXSOCIO C3
4   ON C4.CODIGO = C3.COOPE
5   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:

```

1 cursor footer is
2   SELECT C1.NOMBRE
3   FROM COOPERATIVA C1
4   LEFT JOIN COOPEXSOCIO C2
5   ON C1.CODIGO = C2.COOPE AND C2.SOCIO = codigoSocio
6   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 a cabo 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).