



Escuela de Informática

Diseño e implementación de bases de datos

Proyecto de Integración y Mayorización de Asientos Contables en Oracle mediante

Database Links

Víctor Quesada Rodríguez

Manuel Mora Sandi

Santiago Solís Bolaños

Nicole Vivas Montero

Profesor: MS.c. Johnny Villalobos Murillo

I Ciclo, 2024

Introducción:

Hoy en día, es una realidad que la mayoría de empresas manejan su información más valiosa de manera digital por medio de bases de datos y dentro de estas existe información invaluable. En un mundo donde predominan los negocios, la contabilidad es un componente fundamental para cualquier empresa que desee ser competitiva.

Se nos presenta un problema que se debe resolver utilizando conocimientos de contabilidad junto con la materia que hemos recibido durante el curso de Bases de Datos. El proyecto trata acerca de la automatización del proceso de integración y mayorización de asientos contables de una empresa con bases de datos distribuidas en diferentes lugares.

La finalidad del proyecto es aplicar los conocimientos aprendidos durante el curso hasta el momento e investigar acerca de temas de contabilidad, utilizando nuestro razonamiento para así dar con una solución robusta. Se busca que el proyecto nos acerque a una posible realidad laboral, donde se trabaja con lo más valioso de una empresa.

Este proyecto nos brinda una experiencia académica de gran peso, pues nos prepara para situaciones fuera de la universidad, brindando seguridad y experiencia a la hora de trabajar.

Herramientas utilizadas:

El proyecto se llevará a cabo utilizando una herramienta utilizada por muchas empresas importantes, Oracle SQL Developer. Es una herramienta potente y segura, llena de muchas funcionalidades. Nos permitirá almacenar toda la información necesaria en tablas, además de los procedimientos almacenados que nos servirán para automatizar los procesos de mayorización e integración de asientos contables.

Inicialmente, se trabajará con tres tablas principales: asientos contables, catálogo contable y errores. En estas tablas se insertará la información con la que trabajará la empresa ficticia. La tabla de asientos contables contará con un número de asiento, una cuenta, cuenta que afecta, un debe (débito) y un haber (crédito). Cada fila de esta tabla representa un asiento contable que registra un movimiento entre las cuentas especificadas.

La tabla del catálogo contable hace referencia a un catálogo de cuentas contables, su función es validar las cuentas utilizadas en los asientos contables y establecer montos. Los campos de esta tabla son: número de cuenta, montos de “debe” y “haber”, además de un valor que nos dice si la cuenta acepta movimientos.

Por último, la tabla de errores se utiliza para registrar errores encontrados cuando se procesan los asientos contables. Tiene dos campos: cuenta (asociada al error) y mensaje (descripción del error). Su función es registrar los problemas conforme vayan surgiendo. Existen varios tipos de errores, como por ejemplo: cuenta no

existe, cuenta no acepta movimientos, debe/haber incorrectos, la suma del debe y haber debe cuadrar. Es importante recalcar que se llevan a cabo validaciones importantes, estas ayudan a mantener integridad en la base de datos, siendo así más precisa.

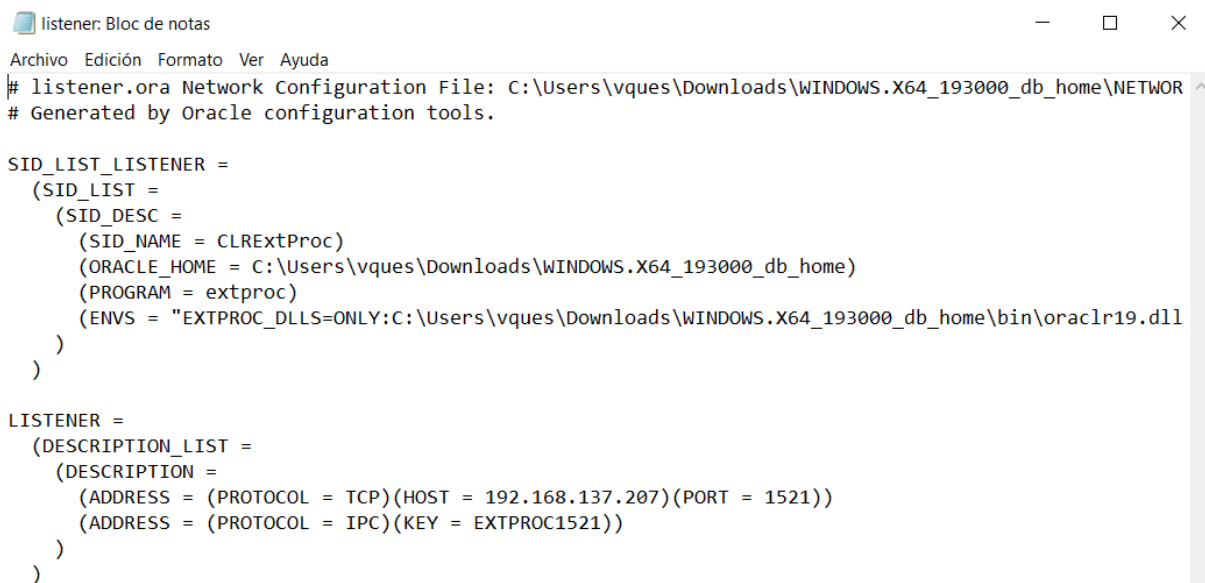
Otra herramienta utilizada que forma parte de los requerimientos funcionales del proyecto es el uso del Oracle Database Link. Esta herramienta integrada nos permite crear y establecer conexiones entre varias bases de datos distribuidas. El DB Link nos permite conectarnos por medio de la red para generar consultas y correr procedimientos, dando la impresión de que todos trabajan con la misma base de datos, aunque sean múltiples y en diferentes localidades.

Dentro de la base de datos existen varios procedimientos almacenados que nos permiten llevar a cabo los procesos solicitados. Existen procedimientos de verificación de cuentas, movimientos, del debe y haber y una verificación de asientos, siendo este último uno de los procedimientos más importantes. También existen dos procedimientos de mayorización que serán explicados más adelante.

Se utilizó de igual manera la librería propia de Oracle “UTL_FILE” para documentar los errores en un archivo de texto como un registro que no sería eliminado al hacer un rollback. Más adelante se explicará a detalle.

Consultas SQL y procedimientos utilizados para Database Link:

En este apartado se discutirá el proceso que se llevó a cabo para poder utilizar la herramienta del DB Link. Se utilizaron varias máquinas con Oracle XE, Oracle SQL Developer y conexión a red para acceder a los datos. Primeramente, se debe configurar el listener de la siguiente manera:




```
listener: Bloc de notas
Archivo Edición Formato Ver Ayuda
# listener.ora Network Configuration File: C:\Users\vques\Downloads\WINDOWS.X64_193000_db_home\NETWORK
# Generated by Oracle configuration tools.

SID_LIST_LISTENER =
  (SID_LIST =
    (SID_DESC =
      (SID_NAME = CLRExtProc)
      (ORACLE_HOME = C:\Users\vques\Downloads\WINDOWS.X64_193000_db_home)
      (PROGRAM = extproc)
      (ENVS = "EXTPROC_DLLS=ONLY:C:\Users\vques\Downloads\WINDOWS.X64_193000_db_home\bin\oraclr19.dll"
    )
  )
)

LISTENER =
  (DESCRIPTION_LIST =
    (DESCRIPTION =
      (ADDRESS = (PROTOCOL = TCP)(HOST = 192.168.137.207)(PORT = 1521))
      (ADDRESS = (PROTOCOL = IPC)(KEY = EXTPROC1521))
    )
  )
)
```

Seguidamente se busca el archivo de configuración en Oracle, se cambia el host y se agrega la dirección de red de la máquina para que ahora el listener funcione por medio de la red. Ahora, pasamos al archivo “tnsnames”.

Ya sobre el archivo, se debe cambiar el localhost por la dirección de red, igual que el paso anterior (en este caso la dirección de la máquina es 192.168.137.207 en ORCL y ORCLPDB). En los servicios, el ORCL es el que trae por defecto, se debe agregar el servicio ORCLPDB con las mismas especificaciones.

 tnsnames: Bloc de notas

Archivo Edición Formato Ver Ayuda

```
# tnsnames.ora Network Configuration File: C:\Users\vques\Downloads\WINDOWS.X64_193000_c  
# Generated by Oracle configuration tools.
```


```
LISTENER_ORCL =  
  (ADDRESS = (PROTOCOL = TCP)(HOST = 192.168.137.207)(PORT = 1521))
```

```
ORACLR_CONNECTION_DATA =  
  (DESCRIPTION =  
    (ADDRESS_LIST =  
      (ADDRESS = (PROTOCOL = IPC)(KEY = EXTPROC1521))  
    )  
    (CONNECT_DATA =  
      (SID = CLRExtProc)  
      (PRESENTATION = RO)  
    )  
  )
```

```
ORCL =  
  (DESCRIPTION =  
    (ADDRESS = (PROTOCOL = TCP)(HOST = 192.168.137.207)(PORT = 1521))  
    (CONNECT_DATA =  
      (SERVER = DEDICATED)  
      (SERVICE_NAME = orcl)  
    )  
  )
```

```
ORCLPDB =  
  (DESCRIPTION =  
    (ADDRESS = (PROTOCOL = TCP)(HOST = 192.168.137.207)(PORT = 1521))  
    (CONNECT_DATA =  
      (SERVER = DEDICATED)  
      (SERVICE_NAME = orclpdb)  
    )  
  )
```

A continuación nos ubicamos en el Símbolo del Sistema y ejecutamos un comando para detener el listener, a su vez se vuelve a iniciar para la actualización del Oracle. Antes de crear la base de datos, es importante recalcar que se debe ingresar al SQL Plus para crear un usuario con privilegios para el DB Link. Cuando estamos creando la base de datos se coloca el nombre y contraseña del usuario con acceso, en el campo de host se debe colocar la dirección IP de la máquina, el puerto de Oracle es el estándar y el nombre del servicio es el agregado en el archivo TNS. Se da click al botón probar y en caso de ser correcto se guarda y podemos dar click a conectar para comenzar a trabajar.

Name  Color

Tipo de Base de Datos

Información de usuario Usuario de Proxy

Tipo de autenticación

Usuario Rol

Contraseña ☒ Guardar Contraseña

Tipo de Conexión

Detalles Avanzado

Nombre del Host

Puerto

☐ SID

☒ Nombre del Servicio

Ya estando dentro, para realizar acciones es necesario utilizar los comandos de la siguiente manera:

```
CREATE PUBLIC DATABASE LINK DBLINK CONNECT TO usuario1
IDENTIFIED BY Victor USING ' (DESCRIPTION = (ADDRESS =
(PROTOCOL = TCP)(HOST = 192.168.137.207)(PORT = 1521))
(CONNECT_DATA = (SERVER = DEDICATED) (SERVICE_NAME = orclpdb)
) )';
```

```
INSERT INTO tabla_asientos@DBLINK (numero, cuenta, centaAfecta, debe,
haber) VALUES (1, '99-99-99', '22-02-02', 400, 0);--pago a planilla
commit;
```

```
INSERT INTO tabla_asientos@DBLINK (numero, cuenta, centaAfecta, debe,
haber) VALUES (1, '11-01-01', '22-02-02', 500, 500);--pago a planilla

commit;
```

Creación de tablas:

- 1) **tabla_asientos:** esta tabla incluye con un número de asiento, una cuenta, cuenta que afecta, un debe (débito) y un haber (crédito). Cada fila de esta tabla representa un asiento contable que registra un movimiento entre las cuentas especificadas.

```
CREATE TABLE tabla_asientos(  
    numero INT,  
    cuenta VARCHAR(20),  
    centaAfecta VARCHAR(20),  
    debe FLOAT,  
    haber FLOAT  
);
```

- 2) **catalogo_contable:** esta tabla hace referencia a un catálogo de cuentas contables, su función es validar las cuentas utilizadas en los asientos contables y establecer montos. Los campos de esta tabla son: número de cuenta, montos de “debe” y “haber”, además de un valor que nos dice si la cuenta acepta movimientos.

```
CREATE TABLE catalogo_contable(  
    cuenta VARCHAR(20),  
    acepta_movimiento CHAR(1),  
    debe FLOAT,  
    haber FLOAT  
);
```

- 3) **errores:** se utiliza para registrar errores encontrados cuando se procesan los asientos contables. Tiene dos campos: cuenta (asociada al error) y mensaje (descripción del error). Su función es registrar los problemas conforme vayan surgiendo. Existen varios tipos de errores, como por ejemplo: cuenta no existe, cuenta no acepta movimientos, debe/haber incorrectos, la suma del debe y haber debe cuadrar.

```
CREATE TABLE errores (  
    cuenta VARCHAR(20),  
    mensaje VARCHAR(255)  
);
```


Procedimientos y funciones almacenadas:

- 1) **verificarExistenciaCuenta:** este procedimiento verifica si una cuenta existe o no en la tabla del catálogo contable. Este procedimiento recibe dos parámetros: p_cuenta (cuenta a verificar) y p_resultado (un parámetro de salida, indica el resultado de la verificación).
Dentro del procedimiento se declara vCount que almacenará el resultado del recuento de filas. Se inicia el proceso y se comprueba que si el valor de vCount es igual a 0, entonces habrá un error y p_resultado será 1, además se insertará un error en la tabla de errores explicando que la cuenta no existe. En caso de encontrar la cuenta p_resultado será 0 y dará fin al procedimiento. Se incluye una excepción que insertará un error genérico de ser necesario

```
CREATE OR REPLACE PROCEDURE verificarExistenciaCuenta(p_cuenta IN VARCHAR2, p_resultado OUT NUMBER)
IS
    vCount NUMBER;
BEGIN
    SELECT COUNT(*) INTO vCount FROM catalogo_contable WHERE cuenta = p_cuenta;
    IF vCount = 0 THEN
        p_resultado := 1; -- Si la cuenta no existe, devuelve 1
        INSERT INTO errores(cuenta, mensaje)
        VALUES (p_cuenta, 'La cuenta no existe en el catálogo contable');
    ELSE
        p_resultado := 0; -- Si la cuenta existe, devuelve 0
    END IF;
EXCEPTION
    WHEN OTHERS THEN
        p_resultado := 1; -- En caso de error, devuelve 1
        INSERT INTO errores(cuenta, mensaje)
        VALUES (p_cuenta, 'Error genérico: ' );
END;
/
```

a

- 2) **verificarMovimiento:** este procedimiento almacenado tiene como finalidad ver si se permite o no hacer un movimiento en una cuenta específica, utilizando el catálogo contable.
Al igual que el procedimiento anterior, recibe una cuenta y un resultado como parámetros. Se crea una variable llamada vAcepta y guarda el valor de la columna “acepta movimiento” de “catalogo_contable”. Se hace un select y se busca encontrar la cuenta.
Podemos encontrar la cuenta y que esta no acepte movimientos, en este caso se genera un error. Si esta acepta movimientos se acaba el procedimiento. Existen otros casos a tomar en cuenta como puede ser que no se encuentre la cuenta o que haya mas de una cuenta encontrada, en estos casos se debe generar un error respectivo. De pasar algo fuera de estos casos descritos se debe generar un error genérico.

```

CREATE OR REPLACE PROCEDURE verificarMovimiento(p_cuenta IN VARCHAR2, p_resultado OUT NUMBER)
IS
    vAcepta CHAR(1);
BEGIN
    SELECT acepta_movimiento INTO vAcepta FROM catalogo_contable WHERE cuenta = p_cuenta AND ROWNUM = 1;
    IF vAcepta = 'N' THEN
        p_resultado := 1; -- Si el movimiento no es permitido, devuelve 1
        -- Insertar el error en la tabla de errores
        INSERT INTO errores(cuenta, mensaje)
        VALUES (p_cuenta, 'Movimiento no permitido para esta cuenta');
    ELSE
        p_resultado := 0; -- Si el movimiento es permitido, devuelve 0
    END IF;

EXCEPTION
    WHEN NO_DATA_FOUND THEN
        p_resultado := 1; -- Si no se encuentra la cuenta, devuelve 1
        INSERT INTO errores(cuenta, mensaje)
        VALUES (p_cuenta, 'No se encontró la cuenta en el catálogo contable');
    WHEN TOO_MANY_ROWS THEN
        p_resultado := 1; -- Si hay múltiples registros, devuelve 1
        INSERT INTO errores(cuenta, mensaje)
        VALUES (p_cuenta, 'Múltiples cuentas encontradas en el catálogo contable');
    WHEN OTHERS THEN
        p_resultado := 1; -- En caso de error, devuelve 1
        INSERT INTO errores(cuenta, mensaje)
        VALUES (p_cuenta, 'Error genérico: ');
END;
/

```

- 3) **verificarDebeHaber:** el procedimiento debe verificar los valores del haber y debe para una cuenta en la tabla de asientos. A grandes rasgos, recibe una cuenta y revisa que no haya inconsistencias en estos valores. Si ambos valores son 0 o ambos son diferentes de 0 se generará un error. La idea principal es que si hay un valor en debe, el valor de haber sea 0 y viceversa, en este caso el resultado sería correcto (p_resultado sería 0).

```

CREATE OR REPLACE PROCEDURE verificarDebeHaber(p_cuenta IN VARCHAR2, p_resultado OUT NUMBER)
IS
    vDebe NUMBER;
    vHaber NUMBER;
BEGIN
    -- Sumar todos los debe y haber para la cuenta dada
    SELECT SUM(debe), SUM(haber) INTO vDebe, vHaber FROM tabla_asientos WHERE cuenta = p_cuenta;

    -- Verificar que no haya incoherencias entre debe y haber
    IF (vDebe = 0 AND vHaber = 0) OR (vDebe != 0 AND vHaber != 0) THEN
        INSERT INTO errores(cuenta, mensaje)
        VALUES (p_cuenta, 'Incoherencia entre debe y haber');
        p_resultado := 1; -- Si hay incoherencia entre debe y haber, devuelve 1
    ELSE
        p_resultado := 0; -- Si no hay incoherencia, devuelve 0
    END IF;
EXCEPTION
    WHEN OTHERS THEN
        p_resultado := 1; -- En caso de error, devuelve 1
        INSERT INTO errores(cuenta, mensaje)
        VALUES (p_cuenta, 'Error genérico: ' );
END;
/

```

- 4) **verificar_asientos:** realiza verificaciones y actualiza asientos contables de la tabla de asientos. Este procedimiento utiliza un cursor (c_asientos) que recorre los números de asiento en su respectiva tabla. La función del cursor es recuperar números de asiento y almacenarlos en “v_numero_asiento”, cuando no hay más asientos se sale del bucle y posteriormente se cierra el cursor. Este cursor es el que nos permite pasar por cada asiento. Cuando se recorre con el cursor, se recuperan todos los registros asociados a ese número de asiento, se llama a otros procedimientos mencionados en casos anteriores para hacer verificaciones y agregar errores de ser necesario. Con el primer dígito de las cuentas se aplica una regla de negocio según corresponda, analizando las líneas y verificando que todo cierre. Se hace rollback si hay algún error, pero de no ser así se aplica un commit y termina el proceso. Durante este proceso, se utiliza la librería “UTL_FILE” para registrar los errores en el archivo de texto llamado ERRORES.TXT dentro de la carpeta seleccionada por el usuario con permisos necesarios dentro de un directorio registrado previamente en el sistema.

```

CREATE OR REPLACE PROCEDURE verificar_asientos
IS
-- Declaración de un identificador de archivo para el registro de errores
v_file UTL_FILE.FILE_TYPE;
CURSOR c_asientos IS
    SELECT DISTINCT numero FROM tabla_asientos;

v_numero_asiento tabla_asientos.numero%TYPE;
v_error NUMBER := 0;
primer_digito_cuenta1 CHAR(1); -- Declaración de la variable
primer_digito_cuenta2 CHAR(1); -- Declaración de la variable
-- Declaración de la variable asiento para obtener los datos de la tabla_asientos
asiento tabla_asientos%ROWTYPE;
BEGIN
    -- Intenta abrir el archivo para escritura
    v_file := UTL_FILE.FOPEN('USER_DIR', 'ERRORES.TXT', 'W');
    -- Iniciar la transacción
    BEGIN
        -- Abrir cursor de asientos
        OPEN c_asientos;

        -- Recorrer todos los asientos
        LOOP
            FETCH c_asientos INTO v_numero_asiento;
            EXIT WHEN c_asientos%NOTFOUND;

            -- Cursor para recorrer todos los registros con el mismo número de asiento
            FOR asiento IN (SELECT * FROM tabla_asientos WHERE numero = v_numero_asiento) LOOP
                -- Obtener el primer dígito de la cuenta
                primer_digito_cuenta1 := SUBSTR(asiento.cuenta, 1, 1);
                primer_digito_cuenta2 := SUBSTR(asiento.centaAfecta, 1, 1);

                -- Verificar existencia de las cuentas y permisos de movimiento
                verificarExistenciaCuenta(asiento.cuenta, v_error);
                verificarExistenciaCuenta(asiento.centaAfecta, v_error);
                verificarMovimiento(asiento.cuenta, v_error);
                verificarMovimiento(asiento.centaAfecta, v_error);

                -- Aplicar las reglas de negocio según el primer dígito de las cuentas
                CASE
                    WHEN primer_digito_cuenta1 = '1' AND primer_digito_cuenta2 IN ('2', '4') THEN
                        UPDATE catalogo_contable SET debe = debe + asiento.debe WHERE cuenta = asiento.cuenta;
                        UPDATE catalogo_contable SET haber = haber + asiento.debe WHERE cuenta = asiento.centaAfecta;

                    WHEN primer_digito_cuenta1 = '2' AND primer_digito_cuenta2 IN ('1', '2', '3') THEN
                        UPDATE catalogo_contable SET haber = haber + asiento.haber WHERE cuenta = asiento.cuenta;
                        UPDATE catalogo_contable SET debe = debe + asiento.haber WHERE cuenta = asiento.centaAfecta;

                    WHEN primer_digito_cuenta1 = '3' AND primer_digito_cuenta2 IN ('2', '4') THEN
                        UPDATE catalogo_contable SET debe = debe + asiento.debe WHERE cuenta = asiento.cuenta;
                        UPDATE catalogo_contable SET haber = haber + asiento.debe WHERE cuenta = asiento.centaAfecta;

                    WHEN primer_digito_cuenta1 = '4' AND primer_digito_cuenta2 IN ('1', '3', '5') THEN
                        UPDATE catalogo_contable SET haber = haber + asiento.haber WHERE cuenta = asiento.cuenta;
                        UPDATE catalogo_contable SET debe = debe + asiento.haber WHERE cuenta = asiento.centaAfecta;

                    WHEN primer_digito_cuenta1 = '5' AND primer_digito_cuenta2 IN ('2', '4') THEN
                        UPDATE catalogo_contable SET debe = debe + asiento.debe WHERE cuenta = asiento.cuenta;
                        UPDATE catalogo_contable SET haber = haber + asiento.debe WHERE cuenta = asiento.centaAfecta;

                    ELSE
                        v_error := 1;
                        -- Escribe el error en el archivo de texto
                        UTL_FILE.PUT_LINE(v_file, 'Combinación de cuentas no válida para la transacción: ' || asiento.cuenta || ' afecta ' || asiento.centaAfecta);
                END CASE;

                -- Salir del loop si hay un error
                EXIT WHEN v_error <> 0;
            END LOOP;
        END LOOP;
    END LOOP;

```

```

-- Cerrar cursor de asientos
CLOSE c_asientos;

-- Comprobar si hay errores
IF v_error = 0 THEN
    COMMIT;
ELSE
    ROLLBACK;
END IF;
EXCEPTION
    WHEN OTHERS THEN
        -- En caso de error, registra el error en el archivo de texto
        UTL_FILE.PUT_LINE(v_file, 'Error durante la actualización de asientos: ' || SQLERRM);
        ROLLBACK;
END;

-- Cerrar el archivo después de su uso
UTL_FILE.FCLOSE(v_file);
END;
/

```

```

CREATE OR REPLACE DIRECTORY USER_DIR AS 'C:\app\ssoli\product\21c\admin\XE\dpdump';

```

- 5) **mayorizar_subcuentas:** el propósito del procedimiento es sumar los montos de las “subcuentas” hacia las cuentas principales del catálogo contable. Se toman como cuentas principales aquellas que no aceptan movimientos, se realiza una subconsulta cuya función es sumar los montos de las subcuentas que comienzan con el mismo prefijo de las cuentas principales. En caso de no haber subcuentas se retorna un 0. Cuando se termina de recorrer cada subcuenta, los montos sumados se asignan al debe o haber de la cuenta correspondiente.

```

CREATE OR REPLACE PROCEDURE mayorizar_subcuentas IS
BEGIN
    -- Sumar subcuentas a cuentas principales
    UPDATE catalogo_contable c1
    SET
        c1.debe = (SELECT COALESCE(SUM(c2.debe), 0)
                  FROM catalogo_contable c2
                  WHERE c2.cuenta LIKE c1.cuenta || '-%'), --'-%' Toma el prefijo de la cuenta principal
        c1.haber = (SELECT COALESCE(SUM(c2.haber), 0)
                   FROM catalogo_contable c2
                   WHERE c2.cuenta LIKE c1.cuenta || '-%')
    WHERE c1.acepta_movimiento = 'N'; --Solo hace la suma en cuentas que no aceptan movimientos
END;
/

```

- 6) **mayorizar_contabilidad:** suma los saldos de las subcuentas a las cuentas principales, y suma de la cuenta “principal” a la “superior”, siendo este el nivel mas alto. Automatiza la mayorización al igual que el procedimiento anterior. Asegura que se sumen los montos de las subcuentas a las cuentas principales y de las principales a las cuentas superiores. Se itera del nivel

mas bajo al más alto, pasando entre los tres niveles descritos, se realizan sumas cuando es necesario y posteriormente, se actualizan las cuentas correspondientes.

```
CREATE OR REPLACE PROCEDURE mayorizar_contabilidad IS
BEGIN
    -- Mayorizar desde el nivel mas bajo hacia el mas alto
    FOR i IN REVERSE 1..3 LOOP --En este caso son solo 3 niveles posibles
        EXECUTE IMMEDIATE 'BEGIN mayorizar_subcuentas; END;'; --Suma las subcuentas de las cuentas principales en ese momento
    END LOOP;

    -- Despues de actualizar subcuentas, se actualizan cuentas principales
    UPDATE catalogo_contable c1
    SET
        c1.debe = (SELECT COALESCE(SUM(c2.debe), 0) --Se usa coalesce para en caso de que no hayan cuentas entonces se pone 0
        FROM catalogo_contable c2
        WHERE c2.cuenta LIKE SUBSTR(c1.cuenta, 1, 3) || '%'
        AND c2.cuenta != c1.cuenta), --Evita que la cuenta principal se sume a si misma
        c1.haber = (SELECT COALESCE(SUM(c2.haber), 0)
        FROM catalogo_contable c2
        WHERE c2.cuenta LIKE SUBSTR(c1.cuenta, 1, 3) || '%'
        AND c2.cuenta != c1.cuenta)
    WHERE c1.acepta_movimiento = 'N';
END;
/
```

Conclusiones:

Durante el desarrollo de este proyecto nos encontramos con diferentes problemas, cada uno fue solucionado gradualmente con tal de brindar una solución completa. En el aspecto de las bases de datos distribuidas nos tocó investigar utilizando Internet para lograr un funcionamiento correcto, pero al lograrlo nos permitió conocer cómo funciona una base de datos de nivel empresarial con diferentes localidades que reciben cientos de peticiones al día.

Otra conclusión importante a la que se llegó fue la importancia de manejar bien los errores, pues en algún momento del desarrollo del proyecto, al hacer rollback perdimos datos importantes, pero eso nos forzó a investigar maneras de persistencia utilizando comandos y librería. El manejo correcto de estos errores brinda más seguridad a la base de datos, pues garantiza que no se ingresen datos erróneos y a su vez saber dónde está fallando. De la mano con el proceso de errores viene el proceso de validación de datos, pues al hacer una buena validación, la integridad de los datos se mantiene y permite un mejor registro de errores.

Este proyecto nos acercó a la realidad de los procesos contables que se llevan a cabo en empresas, la investigación acerca de temas de contabilidad aclara muchas dudas que pudimos tener, además de un buen trabajo de forma grupal que desarrolla otras habilidades que debemos tener como profesionales.