Proyecto Bases 3

https://github.com/Achalogy/proj-bases-3

Archivo `.sql` final: query.sql

- Miguel Francisco Vargas Contreras `is147208`
- Nicolas Diaz Granados Cano `is147201`
- Sara Rodriguez Urueña `is147206`

Documentación

Modelo de Dominio

Cambios necesarios

En medio del proceso de desarrollo de este proyecto, tuvimos que adaptar nuestra base de datos a los nuevos requerimientos, entre estos cambios tenemos:

1. Se hizo un cambio en el DDL para agregar el valor `totalpuntos` a la tabla de `Miembro`, asi podemos hacer triggers que permitan actualizar este valor.
2. Eliminar el total_impuesto, es un valor calculado, ayuda a la normalización
3. Eliminado el not null del total_compra, ya que los impuestos y las comprasxproducto necesitan un id_compra, es imposible crear la compra primero y luego agregar el total_compra
4. Agregado `descuento` a la tabla `producto`.

5. Agregado `subtotal_detalle` a la tabla `CompraxProducto` para facilitar las actualizaciones y calculo de totales de compra.

Desarrollo del proyecto

Como equipo nos hemos esforzardo en la calidad de la entrega, por lo tanto se realizaron algunas *pruebas unitarias*, para gran parte de los procesos, estas pruebas nos permitieron reducir el tiempo que se gastaba realizando pruebas continuamente con la base de datos.

Nos guiamos por un sistema de backlog, el cual nos permitio asignar tareas a cada uno de los miembros, siempre velando por que cada uno pudiera realizar una tarea de cada tema.

Organización del proyecto

El proyecto se entrega en un comprimido, este comprimido sigue la siguiente estructura:

```
├──src
│   ├──DFisico
│   ├──entrega
│   │   ├──0-cambios
│   │   ├──1-procedimientos
│   │   ├──2-funciones
│   │   ├──3-disparadores
│   │   ├──4-nuevas-modificaciones
│   │   ├──5-funcionalidades-restricciones
│   │   ├──6-ampliando
│   │   └──7-control-notificaciones
│   └──utils
└──tests
    ├──0-cambios
```

```
        ├──1-procedimientos
        ├──2-funciones
        └──3-disparadores
```

En esta estructura podemos ver todo el código fuente en `src/`, el diseño físico de la base de datos se encuentra en `src/DFisico/`, mientras que todo el proyecto, los disparadores, funciones y procedimientos se encuentran en `src/entrega`, las pruebas unitarias de las que hablamos se encuentran en `tests/`. Cada uno de estos archivos sigue la siguiente lista de tareas:

1. procedimientos
2. funciones
3. disparadores
4. nuevas-modificaciones
5. funcionalidades-restricciones
6. ampliando
7. control-notificaciones

Aspectos tecnicos

Todo el proceso fue unido usando el script `merge.ts`, este contiene la lista de archivos y va leyendo cada uno y agregandolo al archivo final `query.sql`.

Para reducir tiempo en las pruebas y ejecución del proyecto, se diseñaron distintos scripts con tareas especificas:

1. `configOracleDb.ts` Este archivo nos permite crear una conexión a la base de datos usando el paquete de npm `oracledb`.

2. `runSql.ts` este archivo se encarga de ejecutar las instrucciones enviadas, así mismo organiza el resultado de la query, y si es el caso, los mensajes enviados a través de `DBMC_OUTPUT`
3. `build.ts` este archivo ejecuta el `query.sql`

Documentación

Comenzamos creando la tabla `comisiondiariacolaborador`, esta tabla nos va a permitir guardar la comisión diaria que gana un colaborador, esto se logra teniendo como llave primaria `id_colaborador` y `fecha`.

```
create table comisiondiariacolaborador (
   id_colaborador number not null,
   fecha          date not null,
   comisionventas number not null,
   primary key ( id_colaborador,
                 fecha ),
   foreign key ( id_colaborador )
      references colaborador ( id_colaborador )
         on delete set null
)
```

Para facilitar el proceso de calculo de una compra, creamos una función `calcular_sub_total`, esta nos devuelve el sub_total de la compra, es decir la suma de todos sus productos.

```
-- Funcion que calcula el sub total de la compra
create or replace function calcular_sub_total (
   p_id_compra compra.id_compra%type
) return numeric is
```

```
    v_sub_total numeric := 0;
begin
    select sum( cantidad * precio ) as valor
      into v_sub_total
      from producto,
           compraxproducto
     where compraxproducto.id_compra = p_id_compra
       and producto.nombre_producto =
compraxproducto.nombre_producto;
    return v_sub_total;
end;
```

Creamos el procedimiento `calcular_comisiones_diarias`, el cual permite calcular y almacenar la comisión diaria de cada colaborador en la tabla `comisiondiariacolaborador`. Primero, el procedimiento obtiene el total de ventas y el porcentaje de comisión de cada colaborador para el día actual. Luego, verifica si ya se ha registrado una comisión para el colaborador en esa fecha: si existe, actualiza el valor de la comisión; si no, inserta un nuevo registro con la comisión calculada.

```
create or replace procedure
calcular_comisiones_diarias as
    comision    number;
    ventas      number;
    yacalculado number := 0;
begin
    for colab in (
       select colaborador.id_colaborador,
              sum(coalesce(
                 total_compra,
                 0
```

```
            )) as total_ventas,
            porcentaje_comision
      from colaborador
      full outer join (
       select *
         from compra
        where trunc(fecha) = trunc(sysdate)
    ) compra
    on colaborador.id_colaborador =
compra.id_colaborador
     group by colaborador.id_colaborador,
              porcentaje_comision
  ) loop
     select count(*)
       into yacalculado
       from comisiondiariacolaborador
      where id_colaborador = colab.id_colaborador
        and fecha = trunc(sysdate)
        and rownum = 1;

     if ( yacalculado > 0 ) then
        update comisiondiariacolaborador
           set
           comisionventas = colab.total_ventas *
( colab.porcentaje_comision / 100 )
         where
comisiondiariacolaborador.id_colaborador =
colab.id_colaborador;
     else
        insert into comisiondiariacolaborador (
           id_colaborador,
           fecha,
           comisionventas
        ) values ( colab.id_colaborador,
                   trunc(sysdate),
                   colab.total_ventas * (
```

```
colab.porcentaje_comision / 100 ) );
      end if;
   end loop;

   commit;
end;
```

El procedimiento `actualizar_valor_puntos` se encarga de actualizar el valor de los puntos en la tabla `puntos`. Primero, verifica que el parámetro `p_valor_puntos` sea mayor o igual a 0. Si el valor es negativo, se muestra un mensaje de error en la consola informando que no se puede asignar un valor menor a 0. Si el valor es válido, elimina los registros actuales en la tabla `puntos` y luego inserta el nuevo valor proporcionado.

```
create or replace procedure
actualizar_valor_puntos (
   p_valor_puntos numeric
) as
begin
   if p_valor_puntos < 0 then
      dbms_output.put_line('No se puede asignar un
valor menor a 0');
   else
      delete from puntos;
      insert into puntos ( valor ) values (
p_valor_puntos );
   end if;
end;
```

El procedimiento `canjear_puntos` permite a un miembro canjear puntos por un producto específico en una cantidad determinada. Primero, obtiene los puntos totales del miembro y el precio del producto solicitado. Luego, consulta el valor monetario de un punto en la tabla `puntos` y calcula los puntos requeridos para el canje usando el precio del producto y la cantidad. Si el miembro tiene suficientes puntos, actualiza su saldo de puntos restando los puntos usados. En caso contrario, lanza un error indicando que los puntos son insuficientes para completar el canje.

```
create or replace procedure canjear_puntos (
   p_id_miembro         in number,
   p_nombre_producto    in varchar2,
   p_cantidad_producto in number
) as
   v_puntostotales     number;
   v_precio_producto   number;
   v_valor_punto       number;
   v_puntos_requeridos number;
begin
    -- Obtener los puntos totales del miembro
   select totalpuntos
     into v_puntostotales
     from miembro
    where id_miembro = p_id_miembro;

    -- Obtener el precio del producto
   select precio
     into v_precio_producto
     from producto
    where nombre_producto = p_nombre_producto;

    -- Obtener el valor de un punto en unidades
```

```
monetarias (de la tabla Puntos)
   select valor
     into v_valor_punto
     from puntos;

    -- Calcular los puntos requeridos para el
canje
   v_puntos_requeridos := FLOOR((
v_precio_producto * p_cantidad_producto ) /
v_valor_punto);

    -- Verificar si el miembro tiene suficientes
puntos para el canje
   if v_puntostotales >= v_puntos_requeridos then
        -- Actualizar los puntos del miembro
restando los puntos usados
      update miembro
         set
         totalpuntos = totalpuntos -
v_puntos_requeridos
       where id_miembro = p_id_miembro;

   else
      raise_application_error(
         -20003,
         'Error: Puntos insuficientes para
realizar el canje.'
      );
   end if;
end;
```

La función `calcular_puntos_compra` calcula la cantidad de puntos que se obtienen al realizar una compra basada en el precio total de la misma (`p_precio`). Primero, obtiene el valor monetario de un punto

desde la tabla `puntos`. Luego, divide el precio total por este valor y aplica `FLOOR` al resultado para obtener un valor entero, que representa los puntos acumulados por la compra.

```
create or replace function calcular_puntos_compra
(
   p_precio numeric
) return numeric is
   v_puntos     numeric := 0;
   v_cantpuntos numeric := 0;
begin
   select valor
     into v_cantpuntos
     from puntos
    where rownum = 1;

   v_puntos := floor(p_precio / v_cantpuntos);
   return v_puntos;
end;
```

La función `calcular_impuestos` calcula el total de impuestos aplicables a una compra específica basada en su subtotal (`p_subtotal`). Primero, recibe el `id_compra` y el subtotal. Luego, recorre cada impuesto asociado a la compra en la tabla `impuestoxcompra`, aplicando el porcentaje correspondiente sobre el subtotal y acumulando el valor en `v_imp_calculado`. Finalmente, retorna el monto total de impuestos calculado.

```
create or replace function calcular_impuestos (
   p_id_compra compra.id_compra%type,
   p_subtotal  numeric
```

```
) return numeric is
   v_imp_calculado numeric := 0;
begin
   for impuesto in (
      select porcentaje
        from impuestoxcompra
       where id_compra = p_id_compra
   ) loop
      v_imp_calculado := v_imp_calculado + (
p_subtotal * ( impuesto.porcentaje / 100 ) );
   end loop;

   return v_imp_calculado;
end;
```

La función `obtener_saldo_puntos` calcula y retorna el saldo total de puntos de un miembro específico. Primero, obtiene la suma de los puntos acumulados del miembro, filtrando por el tipo de transacción `'ACUMULAR'`. Luego, calcula los puntos canjeados usando el tipo `'CANJE'`. Finalmente, retorna el saldo, que es la diferencia entre los puntos acumulados y los puntos canjeados.

```
create or replace function obtener_saldo_puntos (
   id_miembro number
) return number is
   puntos_acumulados number := 0;
   puntos_canjeados  number := 0;
begin
    -- Obtener puntos acumulados
   select coalesce(
      sum(total_puntos),
      0
   )
```

```
      into puntos_acumulados
      from txpuntos
     where id_miembro = id_miembro
       and tipo = 'ACUMULAR';

     -- Obtener puntos canjeados
    select coalesce(
       sum(total_puntos),
       0
    )
      into puntos_canjeados
      from txpuntos
     where id_miembro = id_miembro
       and tipo = 'CANJE';

     -- Retornar saldo de puntos
    return puntos_acumulados - puntos_canjeados;
end;
```

La función `verificar_existencias` revisa si hay suficientes existencias de un producto en la cafetería asociada a una compra específica. Primero, obtiene el nombre de la cafetería vinculada al `id_compra`. Luego, consulta el inventario para obtener la cantidad disponible del producto en esa cafetería. Finalmente, compara las existencias con la cantidad solicitada y retorna `true` si hay suficientes unidades, o false si no.

```
create or replace function verificar_existencias (
   p_nombre_producto in varchar2,
   p_id_compra       in number,
   p_cantidad        in number
) return boolean is
```

```
    v_existencias      number;
    v_nombre_cafeteria varchar2(100);
begin
    -- Obtener la cafetería asociada a la compra
    select nombre_cafeteria
      into v_nombre_cafeteria
      from colaborador
     where id_colaborador = (
       select id_colaborador
         from compra
        where id_compra = p_id_compra
    );

    -- Obtener las existencias actuales del
producto en la cafetería
    select existencias
      into v_existencias
      from inventariocafeteria
     where nombre_cafeteria = v_nombre_cafeteria
       and nombre_producto = p_nombre_producto;

    -- Verificar si hay suficientes existencias
    if v_existencias >= p_cantidad then
       return true;
    else
       return false;
    end if;
end;
```

El trigger `actualizar_inventario` se ejecuta antes de insertar un registro en la tabla `compraxproducto` para actualizar el inventario de productos en una cafetería específica. Primero, obtiene la cantidad actual de existencias del producto en la cafetería asociada a la compra y verifica si es suficiente mediante la función

`verificar_existencias`. Si no hay existencias suficientes, lanza un error; de lo contrario, reduce el inventario restando la cantidad comprada del total disponible en la tabla `inventariocafeteria`.

```
create or replace trigger actualizar_inventario
before
   insert on compraxproducto
   for each row
declare
   v_existencias number;
begin
   select existencias
     into v_existencias
     from inventariocafeteria --Toma las
existencias iniciales
    where nombre_cafeteria = (
         select nombre_cafeteria --Selecciona el
nombre de la cafeteria donde se va a reducir el
inventario y a que producto se le reduce
           from colaborador
          where id_colaborador = (
            select id_colaborador --Accede a
traves de ID_Colaborador en compra a la cafeteria
correspondiente
              from compra
             where id_compra = :new.id_compra
         )
      ) --Se encarga de tomar la compra que se
esta referenciando
      and nombre_producto = :new.nombre_producto;
--Selecciona el producto en especifico

   if not verificar_existencias(
      :new.nombre_producto,
```

```
        :new.id_compra,
        :new.cantidad
    ) then
        raise_application_error(
            -20002,
            'Error: No hay suficientes existencias
para completar la compra.'
        );
    end if;


    update inventariocafeteria --Actualiza los
valores
        set
        existencias = existencias - :new.cantidad
     where nombre_cafeteria = (
            select nombre_cafeteria
              from colaborador
             where id_colaborador = (
                select id_colaborador
                  from compra
                 where id_compra = :new.id_compra
            )
        )
        and nombre_producto = :new.nombre_producto;
end;
```

El trigger `actualizar_saldo_puntos` se ejecuta después de una inserción, actualización o eliminación en la tabla `txpuntos`. Su función es actualizar el saldo de puntos del miembro correspondiente en la tabla `miembro` cada vez que haya un cambio en los puntos registrados. Dependiendo de la acción (inserción, actualización o eliminación), calcula la diferencia en los puntos y ajusta el saldo de puntos del

miembro correspondiente. Si se elimina un registro, resta los puntos; si se inserta, los suma; y si se actualiza, ajusta la diferencia entre el nuevo y el antiguo valor de los puntos.

```
create or replace trigger actualizar_saldo_puntos
after
   insert or update or delete on txpuntos
   for each row
declare
   v_diff numeric := 0;
begin
   if deleting then
      v_diff := -old.total_puntos;
   elsif inserting then
      v_diff := new.total_puntos;
   elsif updating then
      v_diff := new.total_puntos -
old.total_puntos;
   end if;

   update miembro
      set
      totalpuntos = totalpuntos + v_diff
    where id_miembro = new.id_miembro;
end;
```

El trigger `calcular_comision_en_venta` se ejecuta después de insertar un `pago` en la tabla pago para calcular y actualizar la comisión del colaborador correspondiente a esa compra. Primero, obtiene el `id_colaborador` asociado a la compra, luego verifica si ya se ha calculado la comisión para ese colaborador en el día actual. Si la comisión ya existe, la actualiza; si no, inserta un nuevo registro con la

comisión calculada. El cálculo de la comisión se basa en el porcentaje definido para el colaborador y el total de las ventas realizadas ese día.

```
create or replace trigger
calcular_comision_en_venta after
   insert on pago
   for each row
declare
   v_comision       number := 0;  -- Iniciar como
0 en caso de que no haya ventas previas
   v_ventas         number := 0;  -- Iniciar como
0 en caso de que no haya ventas previas
   yacalculado      number := 0;
   v_id_colaborador
colaborador.id_colaborador%type;
begin
   select compra.id_colaborador
     into v_id_colaborador
     from compra
    where compra.id_compra = :new.id_compra
      and rownum = 1;

   select count(*)
     into yacalculado
     from comisiondiariacolaborador
    where id_colaborador = v_id_colaborador
      and fecha = trunc(sysdate)
      and rownum = 1;

   select sum(coalesce(
      total_compra,
      0
   )) as valor
     into v_ventas
```

```
      from colaborador,
           compra
     where colaborador.id_colaborador =
v_id_colaborador
       and trunc(fecha) = trunc(SYSDATE);

   select porcentaje_comision
     into v_comision
     from colaborador
    where id_colaborador = v_id_colaborador
      and rownum = 1;

   if ( yacalculado > 0 ) then
      update comisiondiariacolaborador
         set
         comisionventas = (v_comision / 100) *
v_ventas
       where
comisiondiariacolaborador.id_colaborador =
v_id_colaborador;
   else
      insert into comisiondiariacolaborador (
         id_colaborador,
         fecha,
         comisionventas
      ) values ( v_id_colaborador,
                 trunc(sysdate),
                 (v_comision / 100) * v_ventas );
   end if;
end;
```

Las tablas `log_miembro`, `auditoria_compra` y `log_inventario` están diseñadas para registrar cambios y auditorías en diferentes áreas del sistema:

1. `log_miembro`: Esta tabla registra los cambios realizados en la información de los miembros, con columnas para el ID del miembro, el tipo de operación (INSERT, UPDATE, DELETE), la fecha y hora del cambio, el usuario que realizó la modificación, y detalles adicionales sobre el cambio.

2. `auditoria_compra`: Similar a `log_miembro`, esta tabla está dedicada a registrar los cambios realizados en las compras. Almacena el ID de la compra, tipo de operación, fecha, usuario que realizó el cambio y detalles sobre la compra modificada.

3. `log_inventario`: Esta tabla realiza un seguimiento de las modificaciones en el inventario de productos de las cafeterías. Registra el nombre de la cafetería, el nombre del producto, la operación realizada, la fecha del cambio, el usuario que efectuó el cambio y detalles adicionales del inventario.

Estas tablas son esenciales para auditoría y control, permitiendo rastrear y revisar las acciones realizadas en la base de datos.

```
create table log_miembro (
   id_log     number not null
      generated by default on null as identity,
   id_miembro number not null,
   operacion  varchar2(10) not null, -- Puede ser
'INSERT', 'UPDATE' o 'DELETE'
   fecha      timestamp default current_timestamp
not null,
   usuario    varchar2(50), -- Usuario que realizó
el cambio
   detalles   varchar2(4000), -- Información
adicional del cambio
```

```
    primary key ( id_log ),
    foreign key ( id_miembro )
       references miembro ( id_miembro )
          on delete set null
);

create table auditoria_compra (
    id_log    number not null
       generated by default on null as identity,
    id_compra number not null,
    operacion varchar2(10) not null,
    fecha     timestamp default current_timestamp
not null,
    usuario   varchar2(50),
    detalles  varchar2(4000),
    primary key ( id_log ),
    foreign key ( id_compra )
       references compra ( id_compra )
          on delete set null
);

create table log_inventario (
    id_log           number
       generated by default on null as identity
    not null,
    nombre_cafeteria varchar2(50) not null,
    nombre_producto  varchar2(50) not null,
    operacion        varchar2(10) not null,
    fecha            timestamp default
current_timestamp not null,
    usuario          varchar2(50),
    detalles         varchar2(4000),
    primary key ( id_log ),
    foreign key ( nombre_cafeteria )
       references cafeteria ( nombre )
          on delete set null,
```

```
    foreign key ( nombre_producto )
       references producto ( nombre_producto )
          on delete set null
);
```

Los triggers creados permiten registrar los cambios realizados en las tablas `miembro` e `inventariocafeteria` en sus respectivas tablas de log (`log_miembro` y `log_inventario`).

1. `log_miembro_trigger`: Este trigger se activa después de cualquier operación (`insert`, `update`, `delete`) sobre la tabla `miembro`. Dependiendo de la operación, se inserta un registro en la tabla `log_miembro` que incluye el `id_miembro`, la operación realizada, el usuario que ejecutó el cambio y un mensaje detallando el cambio realizado.

2. `log_inventario_trigger`: Similar al anterior, este trigger se ejecuta después de las operaciones sobre la tabla `inventariocafeteria`. Registra en la tabla `log_inventario` detalles como el nombre de la cafetería, el producto, la operación realizada, el usuario y una breve descripción del cambio.

```
create or replace trigger log_miembro_trigger
after
   insert or update or delete on miembro
   for each row
declare
   operacion varchar2(10);
begin
   if inserting then
      operacion := 'INSERT';
```

```
    elsif updating then
       operacion := 'UPDATE';
    elsif deleting then
       operacion := 'DELETE';
    end if;

    insert into log_miembro (
       id_log,
       id_miembro,
       operacion,
       usuario,
       detalles
    ) values ( log_miembro_seq.nextval,
               :old.id_miembro,
               operacion,
               user,
               'Cambio en la tabla Miembro' );
end;

create or replace trigger log_inventario_trigger
after
    insert or update or delete on
inventariocafeteria
    for each row
declare
    operacion varchar2(10);
begin
    if inserting then
       operacion := 'INSERT';
    elsif updating then
       operacion := 'UPDATE';
    elsif deleting then
       operacion := 'DELETE';
    end if;

    insert into log_inventario (
```

```
        id_log,
        nombre_cafeteria,
        nombre_producto,
        operacion,
        usuario,
        detalles
    ) values ( log_inventario_seq.nextval,
               :old.nombre_cafeteria,
               :old.nombre_producto,
               operacion,
               user,
               'Cambio en la tabla
Inventario_Cafeteria' );
end;
```

El procedimiento `actualizar_valor_total` se utiliza para actualizar el valor total de una compra en la tabla `compra`. Recibe como parámetro el identificador de la compra (`p_id_compra`). Primero, calcula el sub-total de la compra utilizando la función `calcular_sub_total`, que suma el valor de los productos de la compra. Luego, se actualiza el campo `total_compra` con la suma del sub-total y el resultado de la función `calcular_impuestos`, que calcula los impuestos correspondientes. Si el cálculo de impuestos resulta en `null`, se utiliza `0` como valor predeterminado gracias a la función `coalesce`. Esto asegura que el total de la compra siempre se calcule correctamente, incluso si no se pueden calcular los impuestos

```
create or replace procedure actualizar_valor_total
(
    p_id_compra compra.id_compra%type
) as
    v_sub_total numeric := 0;
```

```
begin
   v_sub_total := calcular_sub_total(p_id_compra);
   update compra
      set
      total_compra = coalesce(
         v_sub_total + calcular_impuestos(
            p_id_compra,
            v_sub_total
         ),
         0
      )
    where id_compra = p_id_compra;
end;
```

El procedimiento `actualizar_totales_compras` actualiza el valor total de todas las compras registradas en la tabla `compra`. Realiza un ciclo `for` que recorre todos los identificadores de las compras (`id_compra`) en la tabla `compra`. Para cada `id_compra`, llama a la procedura `actualizar_valor_total`, que recalcula el total de la compra utilizando el sub-total y los impuestos asociados. Esta procedura es útil para mantener actualizados los totales de todas las compras en el sistema de manera eficiente.

```
create or replace procedure
actualizar_totales_compras as
begin
   for cc in (
      select id_compra
        from compra
   ) loop
      actualizar_valor_total(cc.id_compra);
```

```
    end loop;
end;
```

El trigger `actualizar_valor_total_trigger` se activa después de insertar, actualizar o eliminar registros en la tabla `compraxproducto`. Su propósito es actualizar el valor total de la compra (`total_compra`) en la tabla `compra` cada vez que se realiza un cambio en los productos asociados a una compra.

1. En la inserción (`inserting`): Se obtiene el precio del producto insertado y se suma al total de la compra, considerando la cantidad del producto y los impuestos correspondientes.

2. En la actualización (`updating`): Se ajusta el total de la compra restando el valor del producto que se está actualizando (con la cantidad anterior) y sumando el nuevo valor del producto (con la cantidad actualizada), aplicando los impuestos de ambos valores.

3. En la eliminación (`deleting`): Se resta del total de la compra el valor del producto eliminado (con la cantidad correspondiente) y los impuestos relacionados.

```
-- Se actualiza el valor total de la compra,
tenemos en cuenta que se puede aplicar el impuesto
a cada producto individualmente
create or replace trigger
actualizar_valor_total_trigger after
    insert or update or delete on compraxproducto
    for each row
declare
    v_precio numeric := 0;
```

```
begin
   if inserting
   or updating then
      select precio
        into v_precio
        from producto
       where nombre_producto =
:new.nombre_producto;
   else
      select precio
        into v_precio
        from producto
       where nombre_producto =
:old.nombre_producto;
   end if;

   if inserting then
      update compra
         set
         total_compra = coalesce(
            total_compra,
            0
         ) + ( v_precio * :new.cantidad ) +
calcular_impuestos(
            :new.id_compra,
            v_precio * :new.cantidad
         )
       where id_compra = :new.id_compra;
   elsif updating then
      update compra
         set
         total_compra = ( coalesce(
            total_compra,
            0
         ) - ( v_precio * :old.cantidad ) -
calcular_impuestos(
```

```
            :old.id_compra,
            v_precio * :old.cantidad
         ) ) + ( v_precio * :new.cantidad ) +
calcular_impuestos(
            :new.id_compra,
            v_precio * :new.cantidad
         )
       where id_compra = :new.id_compra;
   elsif deleting then
      update compra
         set
         total_compra = coalesce(
            total_compra,
            0
         ) - ( v_precio * :old.cantidad ) -
calcular_impuestos(
            :old.id_compra,
            v_precio * :old.cantidad
         )
       where id_compra = :old.id_compra;
   end if;

end;
```

El trigger `aplicar_descuento` se activa antes de insertar o actualizar registros en la tabla `compraxproducto`. Su objetivo es aplicar un descuento sobre el precio de un producto al agregarlo o modificarlo en una compra, y luego actualizar el total de la compra correspondiente.

Primero, se obtiene el descuento y el precio del producto desde la tabla `producto` para calcular el nuevo subtotal con descuento. Si el descuento es mayor que 0, se aplica al precio según la cantidad del

producto, de lo contrario, se calcula el subtotal sin descuento. El valor del subtotal con descuento o sin descuento se asigna a la columna `subtotal_detalle` del registro en la tabla `compraxproducto`. Luego, se actualiza el total de todas las compras que contienen el producto afectado, recalculando el total de cada compra sumando los subtotales de los detalles de la compra.

```
create or replace trigger aplicar_descuento before
   insert or update on compraxproducto
   for each row
declare
   v_descuento              number;
   v_precio                 number;
   v_subtotal_con_descuento number;
begin
   select descuento,
          precio   --Tomamos de prodcuto para
calcular subtotal
     into
      v_descuento,
      v_precio
     from producto
    where nombre_producto = :new.nombre_producto;

   if v_descuento > 0 then
      v_subtotal_con_descuento := :new.cantidad *
v_precio * ( 1 - ( v_descuento / 100 ) ); --Creo
que se sobreentiende que hace :|
   else
      v_subtotal_con_descuento := :new.cantidad *
v_precio;
   end if;
```

```
    :new.subtotal_detalle :=
v_subtotal_con_descuento; --Establecemos el valor
del subtotal


   update compra --Actualizamos TODAS las compras
que tengan el producto que se determino que tiene
descuento
      set
      total_compra = (
         select sum(cp.subtotal_detalle) --
Obtenemos todos los detalles asociados a la compra
y los sumamos
           from compraxproducto cp
          where cp.id_compra = compra.id_compra
      )
    where id_compra in ( --Obtenemos los detalles
de compra que tengan el producto asociado
      select id_compra
        from compraxproducto
       where nombre_producto =
:new.nombre_producto
   );

end;
```

La tabla `notificaciones_puntos` se utiliza para almacenar notificaciones relacionadas con los puntos de los clientes. Cada registro en la tabla tiene un identificador único (`id_notificacion`) como clave primaria. Además, incluye la fecha y hora de la notificación (fecha), el identificador del destinatario que hace referencia al cliente (miembro) que recibe la notificación (destinatario), y el mensaje de la notificación (mensaje), con un límite de 400 caracteres. Esta estructura

permite registrar y gestionar las notificaciones de puntos de forma eficiente.

```
create table notificaciones_puntos (
   id_notificacion number
      generated by default on null as identity,
   fecha           timestamp,
   destinatario    number not null, -- Referencia
al ID del cliente (ID_Miembro)
   mensaje         varchar2(400),
   primary key ( id_notificacion ),
   foreign key ( destinatario )
      references miembro ( id_miembro ) on delete
set null
);
```

La tabla `notificaciones_puntos` se usa para almacenar notificaciones relacionadas con los puntos de los clientes. El campo `id_notificacion` es una clave primaria autogenerada, que se asigna automáticamente si no se proporciona un valor. La columna `fecha` guarda la fecha y hora de la notificación, mientras que `destinatario` es una referencia al `id_miembro` del cliente, que no puede ser nula, y está vinculado a la tabla `miembro` mediante una clave foránea. Si se elimina un miembro, el campo `destinatario` se establece en `null` gracias a la opción `on delete set null`. Finalmente, el campo `mensaje` almacena el texto de la notificación con un límite de 400 caracteres.

```
create or replace trigger
notificacion_puntos_insuficientes before
   insert on txpuntos
```

```
    for each row
declare
    puntos_acumulados number;
    puntos_necesarios number := :new.total_puntos;
begin
     -- Calcular puntos acumulados del miembro
    select coalesce(
       sum(total_puntos),
       0
    )
      into puntos_acumulados
      from txpuntos
     where id_miembro = :new.id_miembro
       and tipo = 'ACUMULAR';

     -- Verificar si los puntos acumulados son
suficientes para la redención
    if puntos_acumulados < puntos_necesarios then
         -- Insertar notificación de puntos
insuficientes
       insert into notificaciones_puntos (
          id_notificacion,
          id_miembro,
          mensaje
       ) values (
notificaciones_puntos_seq.nextval,
                 :new.id_miembro,
                 'Puntos insuficientes para
redimir el producto. Intento el ' || to_char(
                    sysdate,
                    'YYYY-MM-DD HH24:MI:SS'
                 ) );
    end if;
end;
```