```
-- correction pour les étudiants --

-- question 1.1 : affichage des emprunts
----------------------------------------
-- si le client n'existe pas, renvoie false
-- sinon affiche les emprunts d'un client (s'il en a) et renvoie true
create or replace function empruntsClient(idcli integer)
returns BOOLEAN
AS $$
DECLARE
  le_nom client.nomcli%type;
  le_prenom client.prenomcli%type;
  e record;
BEGIN
    -- on cherche le client idcli
    select nomcli, prenomcli into le_nom, le_prenom from client
    where ncli = idcli ;
    if not found then
      return false ;
    end if ;
    raise notice 'client %', le_nom||' '||le_prenom ;
    -- on cherche ses emprunts
    -- D'après le message à afficher, on a besoin de Emprunt.montant et de Compte.nag.
    -- Pour relier le compte au client, on a besoin de Compte_client. Les autres tables sont inutiles.
    for e in select nemprunt, ncompte, montant, nag
             from emprunt join compte_client using(ncompte)
             join compte using(ncompte) where ncli=idcli loop
        raise notice 'emprunt numero % de % € sur le compte % de l''agence %',e.nemprunt, e.montant, e.ncompte, e.nag ;
    end loop ;
    -- on regarde si la requete precedente a eu un resultat
    if not found then
      raise notice 'pas d''emprunt';
    end if;
    return true;
END;
$$
LANGUAGE plpgsql ;

-- question 1.2 : vérification du type de compte
------------------------------------------------
-- c'est un trigger qui se déclenche quand on insère ou on met à jour Compte_Client
-- comme c'est un trigger For Each Row, on a une ligne courante dans Compte_Client
-- c'est une ligne que l'on vient d'insérer (valeur NEW) ou modifier (valeurs OLD et NEW, avant et après modif).
CREATE or REPLACE FUNCTION verif_compte() RETURNS trigger AS $$
    DECLARE
      t compte.typecpte%type;
      cl compte_client.ncli%type;
```

```
    BEGIN
        -- on recherche le type du compte dont l'identifiant est donné par NEW
        select compte.typecpte into t
        from compte
        where ncompte = NEW.ncompte ;
        raise notice 'type de ce compte : %',t ;
        -- si c'est un compte courant on peut avoir plusieurs propriétaires
        -- sinon, on doit en limiter le nombre à 1
        if t != 'cpte courant' then
          -- trigger BEFORE donc la modification que l'on est en train de faire sur la ligne courante
          -- n'est pas encore prise en compte
          select ncli into cl from compte_client
          where ncompte = NEW.ncompte;
          if found then
            raise Exception 'ce compte ne peut être partagé';
            --> on arrête là
          end if ;
        end if ;
        RETURN NEW; --> NEW existe pour update et insert
    END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER trig_verif_compte BEFORE INSERT OR UPDATE ON compte_client
FOR EACH ROW EXECUTE FUNCTION verif_compte();

-- Remarque : pour supprime un trigger, on rappelle le nom de la table
-- DROP TRIGGER <nom_trigger> ON <nom_table> ;

-- question 1.3 : vérification de l'endettement
-----------------------------------------------
-- trigger qui se déclenche quand on insère ou on modifie la table Emprunt.
-- Dans le sujet on ne montre que des insertions,
-- je l'ai fait aussi pour des update (un delete ne va jamais violer la contrainte)
CREATE or REPLACE FUNCTION verif_endettement() RETURNS trigger AS $$
    DECLARE
      solde_cpte compte.solde%type;
      total integer;
      nouveau_montant integer ;
    BEGIN
        -- montant des emprunts actuels pour le compte lié au nouvel emprunt
        select sum(montant), solde into total, solde_cpte
        from emprunt
          join compte using (ncompte)
        where ncompte = NEW.ncompte
        group by ncompte, solde;
        -- Un update avec modification de ncompte est à traiter comme une insertion.
        -- En effet, ça veut dire qu'on a retiré un emprunt d'un compte pour le relier à un nouveau compte.
        -- Peu importe si on a aussi changé le montant,
```

```
        -- on est certain que le montant total pour l'ancien compte n'augmente pas
        if (TG_OP = 'INSERT' or OLD.ncompte <> NEW.ncompte) then
          nouveau_montant = total+NEW.montant ;
        else -- update du montant sur le même compte
          -- on ajoute au montant total la différence entre les valeurs (ancienne/nouvelles) du montant de l'emprunt modifié
          nouveau_montant = total+NEW.montant-OLD.montant ;
        end if ;
        raise notice 'montant total avant cet emprunt : % ; et après : % ',total,nouveau_montant;
        -- on vérifie la contrainte
        if (nouveau_montant > 3*solde_cpte) then
          raise exception 'limite d''emprunt atteinte';
        end if;
        RETURN NEW;
    END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER verif_endettement BEFORE INSERT or UPDATE of ncompte,montant ON emprunt
    FOR EACH ROW EXECUTE FUNCTION verif_endettement();
-- si AFTER, il compte aussi le nouvel emprunt dans le select, donc il faut modifier le code
-- Quand c'est possible, préférer un trigger BEFORE, c'est moins coûteux au niveau transactionnel (coût du rollback)

-- question 1.4
---------------
-- le schéma définit dans COMPTE la colonne : nag integer references agence
-- Comme une clé étrangère peut être NULL, un compte peut être géré par aucune agence
-- il faut simplement rajouter un NOT NULL
Alter table Compte alter column nag set not null;

-- question 1.5
---------------
-- ici c'est plus délicat parce qu'il faut qu'il y ait toujours au moins une ligne dans la table compte_client
-- ça veut dire que la création d'un compte doit se faire avec la création d'une ligne de compte_client
create or replace function nouveauCompte(numcpte compte.ncompte%type,
                                         idag agence.nag%type,
                                         idcli client.ncli%type,
                                         libtypecpte compte.typecpte%type,
                                         soldeinitial compte.solde%type)
returns BOOLEAN
AS $$
BEGIN
    insert into Compte(ncompte, nag, solde, typecpte) values (numcpte, idag, soldeinitial, libtypecpte);
    insert into Compte_Client(ncli, ncompte) values (idcli, numcpte);
    -- grâce aux contraintes d'intégrité référentielles, si le client n'existe pas ou si l'agence n'existe pas
    -- on quitte la fonction avec une exception
    -- Ici, le return ne sert à rien (on a plutôt affaire à une procédure.)
    return true;
END;
$$
```

```
LANGUAGE plpgsql ;

-- Passer par une fonction aussi pour vérifier la suppression sur compte_client
-- On suppose que le client passé en paramètre est bien propriétaire du compte passé en paramètre.
-- On peut améliorer cette fonction en vérifiant les paramètres.
create or replace function supprimerProprioCompte(numcpte compte.ncompte%type,
                                        idcli client.ncli%type)
returns BOOLEAN
AS $$
DECLARE
  nb integer;
BEGIN
    -- on regarde combien le compte a de propriétaires
    select count(*) into nb
    from compte_client
    where ncompte = numcpte ;
    if nb > 1 then -- pas de problème, il restera un propriétaire après cette suppression
      delete from compte_client where ncompte = numcpte and ncli = idcli ;
    else -- il n'y a plus qu'un propriétaire, on refuse la suppression
      raise Exception 'un compte doit avoir au moins 1 propriétaire';
    end if ;
    return true;
END;
$$
LANGUAGE plpgsql ;

-- il faudrait aussi s'assurer qu'un update de compte_client ne rend pas un compte "orphelin"
-- Il peut être logique d'interdire les update sur cette table compte_client
--> utiliser les droits plutôt qu'une solution procédurale. On verra ça au dernier cours.

-- De plus, on peut ajouter la contrainte "on delete cascade" sur la clé étrangère entre compte_client et compte
--> la suppression d'un compte entraine la suppression de toutes les lignes de compte_client qui concernent ce compte

-- on supprime l'ancienne contrainte
alter table compte_client drop constraint compte_client_ncompte_fkey;

-- on définit la nouvelle
Alter table Compte_client add foreign key(ncompte) references compte on delete cascade;

-- avec psql, on peut vérifier la définition des contraintes avec : \d compte_client

-- on peut maintenant supprimer un compte sur lequel il n'y a pas d'emprunt,
-- Le SGBD supprimera les lignes de compte_client qui sont liées à ce compte.
```