# SQL et PL/pgSQL
# Langage de manipulation et de définition des données
# Fonctions et déclencheurs

## 1 Préambule

**Notations.** Les notations utilisées pour les clés sont celles du cours.

**Rappel du schéma de la base de données `vins`.**

```
                                                                    -- Table Viticulteur
Viticulteur(#n_viticulteur, nom, prenom, v_ville)
                                                                          -- Table Vin
-- Définition
Vin(#n_vin, cru, millesime, region, n_viticulteur=>Viticulteur)
-- Contraintes
cle(cru, millesime, n_viticulteur)
millesime ∈]1970, 2012[
                                                                       -- Table Buveur
Buveur(#n_buveur, nom, prenom, b_ville)
                                                                     -- Table Commande
-- Définition
Commande(#n_commande, n_buveur=>Buveur, n_vin=>Vin, c_date, c_qte)
-- Contraintes
cle(n_buveur, n_vin, c_date)
c_qte ∈ {6, 12, 18}
                                                                    -- Table Livraison
Livraison(#(n_commande=>Commande, l_date), l_qte)
```

**Trame des fonctions.** La trame des fonctions à coder est disponible sur `/pub/IBD/tp_plpgsql`.

## 2 Fonctions et déclencheurs

### 2.1 Gestion des stocks de vins

1. Modifiez la table `vin` pour ajouter un champ `stock` correspondant à la quantité de bouteilles disponibles. Cette quantité ne peut être négative.

2. Créez une fonction permettant d'initialiser le stock d'un vin à une quantité donnée. On lève une exception si la quantité est négative.

```
CREATE OR REPLACE FUNCTION f_initialise_stock_vin(num_vin INTEGER, qte INTEGER)
RETURNS void AS $fun$
    BEGIN
                                -- Le cours est en ligne (mais ne sert à rien sans les notes de cours)
                                        -- La doc est ici: http://docs.postgresqlfr.org/9.4/
    END;
$fun$ LANGUAGE plpgsql;

SELECT f_initialise_stock_vin(170, 12);                                    -- Exécution de la fonction
```

3. Créez une fonction permettant d'initialiser le stock à 24 bouteilles pour tous les vins.

```
CREATE OR REPLACE FUNCTION f_initialise_stock() RETURNS void AS $fun$
    DECLARE
                                                                            -- Probablement un curseur
    BEGIN
          -- Probablement une boucle sur le curseur permettant d'insérer le stock dans la table vin

                                    -- Un appel à la fonction f_initialise_stock_vin serait bienvenu
                                                        -- EXECUTE f_initialise_stock_vin(...)
    END;
$fun$ LANGUAGE plpgsql;

SELECT f_initialise_stock();                                               -- Exécution de la fonction
```

4. Créez un déclencheur qui vérifie, avant qu'une commande ne soit validée, que la quantité de vin disponible est suffisante. Dans le cas contraire, une exception doit être levée.

```
CREATE OR REPLACE FUNCTION f_verifie_stock() RETURNS TRIGGER AS $fun$
...
$fun$ LANGUAGE plpgsql;

CREATE TRIGGER d_verifie_stock ...
EXECUTE PROCEDURE f_verifie_stock();

...                                                                        -- Tests du déclencheur
```

5. Reprenez le déclencheur précédent en diminuant également la quantité de vins disponible chaque fois qu'un vin peut être commandé.

```
DROP TRIGGER d_verifie_stock ON commande;

CREATE OR REPLACE FUNCTION f_diminue_stock() RETURNS TRIGGER AS $fun$
...
$fun$ LANGUAGE plpgsql;

CREATE TRIGGER d_diminue_stock ...
EXECUTE PROCEDURE f_diminue_stock();

...                                                                        -- Tests du déclencheur
```

## 2.2 Calcul d'attributs dérivés

Nous avons vu en TD que la clé primaire de la relation `vin` contenait l'information contenant le millésime. Nous supposons que les quatre premiers caractères de l'attribut `n_vin` constituent le millésime. Les questions 2 et 3 sont indépendantes.

1. Créez un « miroir » de la table `vin`, que l'on appellera par exemple `vin_m`, dont le schéma et les contraintes sont les mêmes que pour `vin`, à l'exception de l'attribut `n_vin` qui sera de type `TEXT`. Vous pouvez réutiliser une partie du script `create_vins.sql` du précédent TP.

2. Créez une fonction, qui pour chaque tuple présent dans la table `vin`, remplit la table `vin_m` avec les informations correspondantes en prenant soin d'intégrer à la clé primaire l'information concernant le millésime.

Pour construire la clé de `vin_m`, on part des hypothèses suivantes, actuellement vérifiées dans notre base de données :
— le numéro du vin est un entier codé sur trois chiffres,
— le millesime est un entier codé sur quatre chiffres.
On souhaite donc que l'attribut `n_vin` de `vin_m` soit une chaîne de caractères qui contient le millésime puis le numéro du vin. Par exemple, le tuple de la table `vin` : `(170, 'BOUZY', 1994, 'CHAMPAGNE', 15)` deviendra dans la table `vin_m` : `('1994170', 'BOUZY', 1994, 'CHAMPAGNE', 15)`.

```
CREATE OR REPLACE FUNCTION f_initialise_vins_m() RETURNS void AS $fun$
    DECLARE
                                                        -- Probablement un curseur, entre autres
    BEGIN
         -- Probablement une boucle sur le curseur permettant de récupérer les informations voulues
                                              -- L'insertion des informations dans la table vin_m

             -- La fonction permettant de convertir un entier en chaîne de caractère est 'to_char'
                                                             -- Par exemple to_char('42', 'FM00')
                                                        -- L'opérateur de concaténation est '||'

                                  -- Pour mémoire, la doc est ici: http://docs.postgresqlfr.org/9.4/
             -- Et bien sûr la plupart des réponses à vos questions sont là: http://www.google.com
    EXCEPTION
           -- Si on appelle la fonction alors que vin_m contient déjà des valeurs (unique_violation)
    END;
$fun$ LANGUAGE plpgsql;
```

3. Créez un déclencheur qui, avant chaque insertion dans `vin_m`, vérifie si la valeur de `millesime` correspond bien au code entré dans `n_vin`. Si c'est le cas ou si `millesime` est `NULL`, on insère le tuple. Dans le cas contraire, on n'insère pas le tuple.

```
CREATE OR REPLACE FUNCTION f_calcule_millesime() RETURNS TRIGGER AS $fun$
              -- La fonction permettant de récupérer une sous-chaîne de caractères est 'substring'
                                           -- Par exemple substring('toto' from 1 to 2) donne 'to'
                                      -- Pour convertir une chaîne en entier on utilisera 'CAST'
                                                        -- Par exemple CAST('42' AS INTEGER)
$fun$ LANGUAGE plpgsql;

CREATE TRIGGER d_calcule_millesime ...
EXECUTE PROCEDURE f_calcule_millesime();
```