**R3.07 - SQL dans un langage de programmation**
**2023/2024**

**TD/TP 4 : Banque (suite)**

**Note: Ce TP a pour objectif de modifier la structure d'une base de données suite à des changements dans le modèle. Nous évaluerons l'impact sur la gestion des contraintes, mettrons à jour les déclencheurs (triggers), et explorerons l'automatisation d'une opération avec des fonctions et des procédures.**
**Assurez-vous de regrouper les scripts à la fin du TP sous forme d'un fichier compressé (.zip). Vous devez soumettre votre travail via l'espace de rendu dédié au TP sur Moodle, correspondant à votre groupe.**

## Vue et insertion

1. Dans un fichier `triggers_vue.sql`, rendre l'insertion dans la vue `Compte_Client` possible. Elle doit générer des insertions dans les tables `Compte` et `Appartient`
2. Ajouter le compte suivant à la vue `Compte_Client` : `numCompte=21, typeCompte=EPARGNE, solde=50, numClient=1`

## Modification de tables et de contraintes

Nous allons procéder à une modification dans la manière de stocker les directeurs des agences. Désormais, le directeur ne sera plus représenté par un booléen dans la table `Agent`, mais par une **clé étrangère** dans la table `Agence`. Cette modification entraîne une double dépendance fonctionnelle entre les tables `Agent` et `Agence`. Il est important de gérer cette dépendance lors de la création/destruction des tables, ainsi que lors de l'insertion/suppression de lignes dans ces tables.

3. Dans le script `tables_agent_agence.sql`, sans modifier le script de création des tables, ajuster la structure des tables `Agent` et `Agence`, en conservant les directeurs des agences
4. Comment mettre en œuvre les contraintes suivantes :
   - Chaque agence doit obligatoirement avoir un directeur
   - Un agent ne peut être désigné comme directeur que pour une seule agence

   Dans le script `tables_agent_agence.sql`, supprimer l'ancien déclencheur (trigger) qui garantissait qu'une agence avait forcément un directeur
5. Pour gérer la contrainte **« Un directeur travaille forcément dans l'agence qu'il dirige »** écrire dans le fichier `triggers_ligne.sql` les déclencheurs (triggers) suivants :
   - Un déclencheur d'état, `trig_Agence_agDir`, qui attribue automatiquement un agent à une agence dès qu'il est nommé directeur de celle-ci
   - Un déclencheur de ligne, `trig_Agent_agDir`, qui interdit la modification de l'agence d'un directeur
6. Afin de prendre en charge la contrainte selon laquelle **« Le directeur d'une agence est mieux rémunéré que les agents de son agence »** :

- Écrire un déclencheur de ligne nommé `trig_Agence_salDir` pour gérer la modification du salaire d'un directeur dans la table `Agence`
- Élaborer un déclencheur d'état nommé `trig_Agent_salDir` pour gérer les deux types de modifications dans la table `Agent`

7. Tester les triggers à l'aide du script de test disponible sur Moodle
8. Modifier les parties du script `problemes.sql` impactées par ce changement.

**Un compte doit avoir au moins un client**

9. Dans le fichier `triggers_etat.sql`, ajouter deux déclencheurs d'états pour vérifier qu'une modification ou suppression ne laisse pas un compte sans client
10. Ajouter également dans le fichier `triggers_etat.sql` un déclencheur qui interdit l'insertion dans la table Compte
11. Tester les triggers à l'aide du script de test disponible sur Moodle
12. Dans un fichier `insert_compte_client.sql`, définir une procédure `insert_compte_client` permettant d'insérer un élément dans la vue `Compte_Client`. Ensuite, tester cette procédure
13. Exécutez le script de remplissage `remplissage_bis.sql`, suivi du fichier `problemes.sql`. Y a-t-il toujours des problèmes après l'exécution de ces scripts ?

**Une demande de virement**

Nous souhaitons mettre en place la gestion d'une demande de virement interne à la banque, mais entre des clients éventuellement différents :

- Si le solde est insuffisant, le virement est refusé, et un message d'erreur est généré
- Si le montant dépasse `1000` euros, le virement est refusé, et un message d'erreur est généré précisant le nom de l'agent qui gère le compte
- Sinon, le virement est réalisé et modifie la base de données en conséquence

14. Écrire une fonction nommée `fct_estinsuffisant` avec les spécifications suivantes:
    - `Entrées` : le numéro de compte et le montant.
    - `Sortie` : `1` si le solde est suffisant et `0` sinon.

15. Écrire une procédure nommée `proc_ajoutOperation` avec les spécifications suivantes :
    - `Entrées` : le type d'opération, le montant, le client, et le compte
    - `Actions` : insère une ligne dans la table Opération et modifie le compte en conséquence

16. Programmer et tester la procédure de virement à l'aide du script disponible sur Moodle