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

TD/TP 3 : Banque (suite)

Note: L'objectif de ce TP est d'utiliser les déclencheurs de lignes et de tables pour assurer la satisfactions des contraintes complexes ainsi que d'implémenter l'auto-incrémentation à l'aide d'un déclencheur et d'une séquence.
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.

Problèmes

1. Dans le script `problemes.sql`, ajouter :
   a) Une procédure anonyme permettant d'afficher l'ensemble des clients partageant le même nom que leurs agents

   Résultat attendu :
   ```
   Tout est bon!
   ```

   b) Une procédure anonyme permettant de visualiser tous les retraits réalisés par des clients sur des comptes dont ils ne sont pas propriétaires

   Résultat attendu :
   ```
   Le client 9 a effectué un retrait sur un compte qui ne lui appartient pas.
   ```

Déclencheurs de lignes

2. Écrire le script `triggers_ligne.sql` permettant de créer les quatre triggers de lignes vérifiant les contraintes suivantes :
   - Une augmentation de salaire ne doit pas dépasser `10%` et une baisse `8%`
   - Un client ne peut pas être conseillé par un agent portant le même nom que lui
   - Un client ne devrait pas être en mesure d'effectuer un retrait dont le montant excède le solde disponible. Le montant retiré doit être déduit automatiquement du solde. Il est essentiel de prendre en compte les scénarios où le montant ou le numéro de compte sont modifiés
   - Un client ne doit pouvoir retirer de l'argent que sur un compte qui lui appartient

   Utiliser le script `tests_triggers_ligne.sql`, disponible sur `Moodle`, pour les tester
3. Intégrer dans le script `triggers_ligne.sql`, un déclencheur qui met à jour le solde du compte lorsqu'un dépôt est effectué sur celui-ci.

Déclencheurs d'état

4. Muter l'agent `Fontaine` à l'agence numéro `4`
5. Écrire le script `triggers_etat.sql` permettant de créer les deux triggers d'état vérifiant les contraintes ci-dessous et qui doivent afficher le nombre d'erreurs

- Une agence a forcément un et un seul directeur
- Le directeur d'une agence est mieux payé que les agents de son agence

6. Utiliser le script `tests_triggers_etat.sql`, disponible sur `Moodle`, pour les tester

Auto-incrémentation de la clé primaire

`AUTO_INCREMENT` disponible sous `MySQL` n'avait pas d'équivalent sous `Oracle 11g`. Heureusement, `GENERATED BY DEFAULT AS IDENTITY` est maintenant disponible sous `Oracle 12c`. L'inconvénient est que l'on ne contrôle pas la séquence sous-jacente qui, par exemple, ne peut pas être réinitialisée.

7. A l'aide du code ci-dessous, implémenter l'auto-incrémentation de la clé primaire `numAgence`

```
DROP SEQUENCE seq_Agence ;
CREATE SEQUENCE seq_Agence ;
CREATE OR REPLACE TRIGGER trig_seq_Agence
BEFORE INSERT ON Agence
FOR EACH ROW
WHEN (NEW. numAgence IS NULL)
BEGIN
    SELECT seq_Agence.NEXTVAL INTO :NEW.numAgence FROM DUAL;
END;
```

8. Tester la requête suivante :
```
INSERT INTO Agence(numAgence) VALUES(NULL)
```
Quel est le résultat de cette opération ? Identifier et résoudre le problème éventuel
9. Procéder de la même manière pour les autres clés primaires simples

Tests

10. Relancer le script de remplissage en affectant l'agent numéro `10` à l'agence `4` au lieu de `2`. Puis lancer le script `problemes.sql`. Quels problèmes existent toujours ?