UGA - IUT2 - Département informatique
BUT 1A Semestre 2 - Exploitation d'une BD - 2021/2022

R2.06 - Contrôle du 11 Avril 2022
Durée : 2h - Coeff. 36/60

Vous pouvez consulter vos documents personnels.
Vous pouvez également consulter toute documentation en ligne mais nous vous recommandons fortement de privilégier la documentation officielle postgreSQL correspondant à la version installée sur votre poste:

- **En anglais : https://www.postgresql.org/docs/13/**
- **En français : https://docs.postgresql.fr/13/**

Tout échange est interdit.

Consignes (à lire attentivement)

En début de séance, exécuter la commande suivante :

```
source /users/info/pub/rendus-etudiants/R2.06/scripts-etu/debut-examen
```

Cette commande vous positionne dans un répertoire contenant :

- `reponses.sql` : fichier de réponses à compléter pendant le contrôle;
- `user.txt` : fichier contenant le user et mot de passe postgresql à utiliser pendant le contrôle;
- `camelot.csv` : fichier à utiliser dans postgresql pour créer vos données dans la Section 1.1;
- `create_base.sql` : fichier à exécuter dans postgresql pour finaliser la création des données en fin de Section 1.1.

Pour vous connecter à votre base postgreSQL d'examen, exécuter la commande :

```
psql -h postgres-info -U LOGIN
```

où `LOGIN` est le nom d'utilisateur qui vous est attribué dans le fichier `user.txt`, suivi de son mot de passe.

Attention ! Seul le fichier `reponses.sql` sera évalué (et pas un fichier ayant un autre nom). Les éléments décrits par ce fichier (tables, vues, fonctions et triggers) devront néanmoins bien avoir été créés **dans votre base d'examen** (et non dans votre base de TP). De plus, les expériences de vérification demandées doivent être recopiées avec leur résultat dans ce même fichier (ils seront également évalués).

A propos des jointures : il est attendu que vous utilisiez `NATURAL JOIN` à chaque fois que cela est possible. Par ailleurs, toute jointure inutile (intégrant une table alors que toutes les informations utiles sont déjà présentes dans les autres) sera pénalisée.

Le rendu officiel de votre copie se fait, en fin de séance, en exécutant la commande suivante :

```
source /users/info/pub/rendus-etudiants/R2.06/scripts-etu/fin-examen
```

Votre connexion postgresql sera alors fermée.

0. Préambule

Dans le cadre de ce contrôle, nous vous invitons à travailler sur une base permettant d'enregistrer les hauts-faits des chevaliers de la Table Ronde, et en particulier de leur Quête du Graal. Chaque haut-fait est réalisé par une équipe constituée de chevaliers qui sont alors tous considérés comme équipiers (sans chef). Au fil des hauts-faits réalisés par les chevaliers, leur mérite augmente (le père Blaise met à jour le mérite de chacun au nouvel an, en fonction des hauts-faits réalisés l'année précédente). Enfin, dans leur quête, les chevaliers trouvent malheureusement des graals qui se révèlent être des faux (un seul Graal peut être reconnu authentique, et c'est alors définitif).

1. Tables temporaires ou vues?

1.1. Création des tables

Des informations étaient gérées jusqu'alors avec un tableur. Pour intégrer ces informations, nous allons dans un premier temps les copier dans une table nommée **`camelot`**, puis les répartir dans trois tables nommées **`chevalier`**, **`haut_fait`** et **`participe`** selon le schéma relationnel suivant :

`chevalier`(nom, pays_naissance, merite)
`haut_fait`(id_hf, lieu, nature, jour)
`participe`(#id_hf,#equipier)

Nous complétons ensuite ces informations par la relation suivante :

`graal`(id, authentique, #decouvreur, #id_hf)

> Attention! Cette section 1.1 est une étape préliminaire pour la suite du contrôle, mais qui peut être assez longue à réaliser tout en ne concernant qu'une petite partie du programme évalué. Il est important que vous consacriez du temps à toutes les autres sections de ce contrôle.
>
> Ainsi, si vous n'avez pas terminé cette partie **30 minutes après le début de l'épreuve**, nous vous recommandons de supprimer le contenu de votre base, d'exécuter la commande **`\i create_base.sql`** et de passer sans plus tarder à la suite. Cela vous permettra de réaliser la suite du contrôle avec une base correcte.

1. La table **`camelot`** doit elle être une table permanente ou temporaire? La créer de façon classique (sans requête) avec les attributs suivants:
    - **`nom`** de type **`varchar`**
    - **`pays_naissance`** de type **`varchar`**
    - **`merite`** de type **`numeric`**
    - **`id_hf`** de type **`numeric`**
    - **`lieu`** de type **`varchar`**
    - **`nature`** de type **`varchar`**
    - **`jour`** de type **`date`**
2. Alimenter cette table avec les données contenues dans le fichier `camelot.csv` grâce à la commande postgreSQL **`\copy`**. Afin d'identifer quelles informations peuvent ne pas être présentes et celles qui sont dupliquées, afficher le contenu de cette table dans une fenêtre de taille maximisée de sorte que chaque tuple apparaisse sur une seule ligne. Il n'est pas demandé de recopier cet affichage dans le fichier de réponses.
3. Créer les trois tables **`chevalier`**, **`haut_fait`** et **`participe`** grâce à trois requêtes, puis en ajoutant les contraintes suivantes :
    - un chevalier est identifié par son nom;
    - la valeur par défaut de son mérite est 0;
    - un haut-fait est identifié par `id_hf`;
    - par défaut, le jour d'un haut-fait est le jour de son insertion;
    - la participation d'un chevalier à un haut-fait est identifée par le couple constitué de son nom (référencé dans `participe` par l'attribut `equipier`) et du numéro du haut-fait (référencé dans `participe` par l'attribut `id_hf`).
4. Enfin, créer la table **`graal`** en exécutant la commande **`\i create_base.sql`**. **Attention! Cette commande crée également les autres tables si elles n'existent pas. Vous pouvez donc l'exécuter dans une base vide de tout contenu si vous n'avez pas réussi les questions précédentes!**
   *Remarque: si les tables existent, le script **`create_base.sql`** ne tentera pas de les créer, et n'insèrera pas des tuples déjà présents.*

1.2. Schéma alternatif et complémentaire

5. Construire une vue nommée `equipe_an_passe` qui sélectionne les tuples de `participe` dont l'attribut `id_hf` identifie un haut-fait ayant eu lieu l'an passé.
6. Construire une vue nommée `bilan(heros,nb_hf)` qui indique pour chaque nom de chevalier présent en base, le nombre de hauts-faits auxquels il a participé l'an passé. S'il n'a participé à aucun haut-fait l'an passé, alors son nom doit apparaître avec la valeur 0 pour `nb_hf`. Par ailleurs, les chevaliers doivent être listés dans l'ordre décroissant du nombre de hauts-faits réalisés.
   Indication : utiliser la vue `equipe_an_passe`. Si vous n'êtes pas parvenu à la créer, vous pouvez faire comme si elle existait pour répondre à cette question.
7. Le père Blaise souhaite mettre à jour le mérite des chevaliers en ajoutant au mérite de chacun d'entre eux, le nombre de hauts-faits auxquels il a participé l'an passé. Ecrire une requête utilisant la vue `bilan`, qu'il peut exécuter pour cela. Afficher ensuite les chevaliers par ordre décroissant de leur mérite.
8. Une application tiers souhaite réaliser des requêtes d'interrogation et de mise à jour d'information sur une relation ayant le schéma suivant : **`traces_du_graal`**`(id,authentique,lieu,jour)` où :
   - `id` est l'identifiant d'un graal trouvé, accompagné de son authenticité;
   - `lieu` et `jour` sont le lieu et la date du haut-fait au cours duquel ce graal a été trouvé.

   Cette relation doit être créée comme résultat d'une requête, mais dans une table temporaire ou dans une vue? La créer.

2. Triggers et fonctions simples sur tables et vues

9. Ecrire le trigger et sa fonction associée pour que la suppression de la dernière participation à un haut-fait entraîne automatiquement la suppression de ce haut-fait. Ceci constituant un effet de bord, il est alors nécessaire d'ajouter une notification de déclenchement devant identifier le trigger (son nom et celui de la table associée).
   [BONUS] : faire en sorte que cette notification ne soit affichée que s'il y a effectivement un effet de bord, c'est à dire si un haut-fait est effectivement supprimé.
10. Expériences :
    - afficher les caractéristiques du haut-fait numéro 4;
    - supprimer la participation de `lancelot` à ce haut-fait;
    - vérifier la suppression de ce haut-fait.
11. Ecrire le trigger et sa fonction associée pour que l'on puisse réaliser une mise à jour sur la vue `traces_du_graal`.
12. Expériences :
    - afficher l'état actuel de `traces_du_graal`;
    - mettre à jour cette vue pour que le graal numéro 2 soit découvert sur l'île de Skye;
    - afficher le nouvel état de `traces_du_graal`.

3. Triggers et fonctions pour vérifier des contraintes complexes

13. Une fois qu'un graal a été authentifié comme le vrai Graal (`authentique` est vrai), alors il ne peut plus perdre ce statut. Ecrire un trigger et sa fonction associée pour que toute tentative pour rendre faux un graal authentique soit annulée avec un message expliquant l'annulation. Aucun effet de bord n'étant causé par la fonction, il n'est pas nécessaire d'ajouter une notification de déclenchement.
14. Expériences :
    - afficher les n-uplets de graal;
    - authentifier le premier graal (id=1);
    - afficher à nouveau les graals;
    - rendre faux ce premier graal;
    - corriger le découvreur de ce premier graal pour que ce soit le chevalier **`galaad`**;
    - afficher à nouveau les graals.
15. Par ailleurs, il ne peut y avoir qu'un seul vrai Graal. Ecrire un trigger et sa fonction associée pour annuler toute action qui irait à l'encontre de cela. Là encore, il est nécessaire d'afficher un message expliquant l'annulation, mais comme aucun effet de bord n'est causé par la fonction, il n'est pas nécessaire d'ajouter une notification de déclenchement.
16. Expériences :
    - authentifier le deuxième graal (id=2)