TD 2 Données 1 : MERISE + PostgreSQL

Exo 0 : Introduction et révisions.

Parce que la tendance est de créer des réseaux sociaux, essayons de créer le nôtre : HEIFB (pas très original mais bon... passons). Pour cela, vous allez implémenter la base de données décrite dans le fichier SQL suivant:
https://drive.google.com/file/d/17KUmZ7PtpB6w02lpL_CJBSCmhBjBM9f7/view?usp=sharelink

Pour cet exo “0”, nous allons juste nous rappeler de tout ce qu’on a fait jusque la, avant les clés étrangères...
1.a - Oops! il faut être anglophone si on veut être international : changez le nom de la colonne "nom" de la table user en "last_name".

1.b - Idem : changez le nom de la colonne "prénom" de la table user en "first_name".

1.c - Il y a quelque chose qui cloche : changez le type de la colonne "create_date" de la table user en un type plus approprié.
Syntaxe : ALTER TABLE ... ALTER COLUMN ... TYPE ... USING create_date::date;

Bon, visiblement, l’administrateur de la base de données a fait de très mauvais choix d’implémentation, nous allons corriger tout ça :
2.a - Supprimez la colonne " age " de la table user (attention user est un mot réservé donc il faut le mettre entre guillemets).

2.b - Ajoutez une contrainte " not null " à la colonne " create_date " de la table user.

2.c - Ajoutez une valeur par défaut appropriée à la colonne "is_blocked" de la table user. Sachant que la valeur de cette colonne est :

- “true” si l'utilisateur est bloqué
- “false” si l'utilisateur ne l'est pas

2.d - ajoutez la colonne "update_time" qui est la date de la dernière modification d'un utilisateur.

3 - On est jamais à l'abri des erreurs donc : comment peut-on s'assurer que la valeur de la colonne "update_time" est toujours antérieure la valeur de la colonne "create_date" ?

C'est bien beau, mais notre base de données est encore vide...
4.a - Insérez 2 nouveaux enregistrements dans la table "user" en spécifiant toutes les valeurs de chaque colonne avec "is blocked" étant "true".

4.b - Insérez 2 nouveaux enregistrements dans la table "user" en spécifiant uniquement le strict minimum, c'est-à-dire le moins de colonnes possibles.

4.c - Insérez 2 nouveaux post de votre choix.

C'est tout pour cette première version. Voyons un peu le bilan...
5.a - Affichez la liste des 5 derniers "user" qui ont créés un compte (bloqués ou non), par ordre décroissant de la date de création.

5.b - Affichez la liste des "user" qui ne sont pas bloqués

Exo 1 : Reverse-engineering SQL vers MERISE

Le but de cet exercice est de partir d'une base de données, et de le modéliser : c'est le concept du reverse-engineering, modéliser un code déjà existant.
Pour cela, nous continuons avec notre fameux réseau social, implémentez la deuxième version à l'aide du fichier SQL suivant :
https://drive.google.com/file/d/15jzNgO5D7T8qj1qtTdfDw7Te5xBGr5vx/view?usp=share_link

À cette implémentation on introduit les règles de gestion suivantes. Pour information, une règle de gestion est une contrainte qui s'applique à une action, à une activité ou encore à un processus. Elles nous permettent notamment de comprendre les relations et contraintes qu'on doit placer sur notre base de données, pour les cas présent les voici :

- Un « user » peut poster autant de « post » qu'il le souhaite;
- Un « post » est posté par un seul « user »
- Un « user » peut jouer a beaucoup de « mini games »
- Un « mini games » peut être utilisé par plusieurs « user » au fil du temps

1 - Commençons par avoir une vue d'ensemble : modéliser la base de données à l'aide d'un modèle conceptuel de données (MCD) en précisant les relations et leurs cardinalités

qui manquent entre les entités (vous pouvez utiliser JMerise si ça marche, ou draw.io, même sur papier ça marche...)

2 - Vu que c'est une nouvelle version, il faudra créer une nouvelle base de données : Ré-implémenter une nouvelle version de la base de données avec toutes les colonnes, tables, et clés étrangères qui permettront de faire la relation entre ces tables.

Le plus important dans un réseau social, c'est d'avoir des ami(e)s. Ajoutons cette notion a notre base de données :
3.a - Mettre à jour votre MCD avec la relation réflexive qui permettra à un « user » d'être ami avec d'autres « user »
3.b - Ajoutez cette dernière spécification dans votre implémentation (votre base de données), et insérez trois enregistrements qui disent chacun : qu'un certain utilisateur est ami avec un autre utilisateur.

4. Bon, il est maintenant temps que nos utilisateurs postent quelque chose et qu'ils jouent à des mini-jeux : Insérez dans la base de données les informations suivantes:

1. l' « user » avec l'id 2 a posté : "J'adore trop suivre des Novelas." le 3 janvier 2024
2. l'« user » avec l'id 3 a posté: " J'ai fini tous les exercices de données. " le 5 janvier 2024; ensuite il a encore posté "Aujourd'hui, ce sera le projet de SYS." le 6 janvier 2024
3. les « user » avec les id 2 , 3, 6 ont joué au « mini games » avec l'id 2
4. l'« user » avec l'id 4 joue aux « mini games » avec les id: 1 et 4
5. Les « user » avec les id 2, 6, 12 et 9 sont respectivement amis avec les « user » avec les id 20, 15, 16 et 2.

Pour faire tout ce qui sera plus tard dans notre réseau social, l'inventaire des mini-jeux joués et la liste des amis de chacun sera nécessaire. Nous allons faire quelques requêtes utiles sur la base de données :
5.a - Affichez la liste de tous les « post » en affichant les colonnes "first_name" de son auteur, "body" (du post) et "create_date" (du post).

5.b - Affichez la liste des "user" qui ont plus de 18 ans et ont déjà joué au « mini games » qui a l'id 3, en affichant les colonnes "first_name", "last_name" et "birthday".

5.c - Affichez la liste des « mini_games » auxquels l' "user" qui a l'id 4 a déjà joué. On affichera toutes les colonnes de la table mini_games.

5.d - Affichez l'user qui a écrit le tout premier « post », en affichant les colonnes "first_name", "last_name", "birthday" de l'user , ainsi que le "body" du "post"

5.f - Question difficile : Affichez la liste des amies de l'user 2, en affichant les colonnes "id", "first_name", "last_name" et "birthday" des amies de l'user 2 (et les siens).

Exo 2 : Amélioration du projet

Dans notre version 2 de HEIFB, il nous manquait un élément crucial : les MP ! Nous proposons donc une version 3 où les utilisateurs peuvent s'envoyer des messages dans un groupe de discussion: pour cela , implémentez la base de données du fichier SQL suivant : https://drive.google.com/file/d/1W_dd_QDHN08COPre2dhflByJ0z6SUSQP/view?usp=share_link

1 - Comme d'habitude commençons par la spécification : Modéliser cette base de données à l'aide d'un modèle conceptuel de données (MCD).

Après avoir fait quelques insert, il est temps d'afficher qui s'est inscrit à notre cher réseau social :
2.a - Affichez la liste des "user" (toutes les colonnes) par ordre décroissant en fonction de leurs noms de famille.

2.b - Affichez la liste des 5 premiers "user" qui ont créé un compte (toutes les colonnes).

2.c - Afficher la liste des « user » qui ont des amies (au moins 1) dans un tableau contenant les colonnes suivantes : nom, prénom, âge, et e-mail, et sans afficher de doublon (renseignez vous sur "distinct" sur postgresql).

2.d - Affichez le contenu du premier message écrit par l'user ayant l'id 1 sur le groupe de discussion ayant l'id 1

2.e - On veut promouvoir les mini jeux qui ont du mal à percer : Afficher la liste des « mini games » qui n'ont pas encore été joués par qui que ce soit.

2.f - La base de données étant lourde, il faut bloquer ceux qui ne sont pas du tout actifs : Afficher les "user" inactifs (qui n'ont pas posté et n'ont pas écrit de message) depuis 3 mois qui ne sont pas déjà bloqués.