Requêtage SQL - ESGI - Partiel : World of Mykraft

> "Qu'est-ce que je fais ici ? Je ne veux pas créer des sites Web, je veux faire des jeux vidéo ! SQL est tellement inutile." - Probablement vous, en ce moment

La plupart des applications de la vie réelle que vous avez l'occasion de voir sont des sites web ennuyeux, avec toujours les mêmes tables, les mêmes problématiques, les mêmes modèles, et vous avez juste envie de retourner coder des choses amusantes en C. La vérité est que SQL est un beau langage qui vous permet de faire de l'algèbre relationnelle[1] avec une syntaxe très simple (et pourtant très expressive). Il est utile dans de nombreux domaines, et il serait dommage de le négliger à cause de certains préjugés erronés.

Au diable les sites web, créons un jeu vidéo.

Présentation

World of Mykraft est un jeu de rôle en ligne massivement multijoueur que vous vous apprêtez à créer. Le joueur peut créer un personnage personnalisé, combattre des créatures, accomplir des quêtes et monter en niveau.

La raison d'être du SQL

Il serait vraiment difficile d'essayer de créer le serveur de ce jeu sans utiliser une base de données. Nous ne pouvons pas simplement stocker les données dans des fichiers, car :

- Les données sont dynamiques. Certaines choses comme les quêtes que vous avez complétées ou les objets que vous avez changent tout le tout le temps, et stocker ces informations dans des fichiers signifierait avoir beaucoup d'entrées/sorties sur le disque.
- Vous devez permettre l'accès simultané à la base de données, car votre serveur sera probablement multi-threadé. Il est beaucoup plus facile de faire de l'accès simultané sur des bases de données qui ont été spécialement conçues pour cela que sur des fichiers.
- Du point de vue du développeur, il serait difficile d'exprimer les opérations relationnelles sans utiliser l'algèbre relationnelle, et il faudrait écrire des tonnes de textes passe-partout inutiles pour contourner ce problème.

Vous pourriez utiliser un ORM[2], mais vous vous heurteriez rapidement au problème de l'Object-Relational impedance mismatch[3] : la plupart des problèmes d'algèbre relationnelle ne peuvent pas être décrits simplement à l'aide d'une syntaxe orientée objet. De plus, les ORM sont connus pour générer des requêtes non optimales la plupart du temps, ce qui peut poser un problème dans

certaines applications à haute performance où la vitesse compte vraiment (comme un MMORPG, par exemple).

Maintenant que vous êtes, je l'espère, convaincu de la nécessité d'une bonne base de données pour ce problème, voyons plus en détail comment le jeu sera organisé.

Schéma

Vous pouvez trouver sur MyGES un schéma de la base de données que nous utiliserons pour ce sujet. Il contient 5 tables pour l'instant :

- `creature_template` qui représente une créature abstraite et ses caractéristiques. Il peut y avoir plusieurs "instances" d'une créature, à différentes positions dans le jeu.
- `creature` contient la liste des instances de ces créatures, leur position et leur état actuel.
- `quest` contient la liste des quêtes.
- `character` est la liste des personnages, leur statut et leurs caractéristiques.
- `character_quests` est la liste des quêtes avec lesquelles les personnages ont interagi.

Les commentaires du schéma doivent être suffisamment explicites pour comprendre l'objectif de chaque champ. Si ce n'est pas le cas, n'hésitez pas à demander à votre intervenant !

Pour l'anecdote, cette base de données est adaptée (et bien sûr, très simplifiée) de la base de données du projet MaNGOS[4] qui se veut être un serveur MMO compatible avec World of Warcraft. Oui, les gens utilisent réellement SQL pour leurs serveurs de jeux.

Exercices

Quelques statistiques

Votre serveur fonctionne depuis quelques jours, faisons quelques statistiques !

- Dans `ex01.sql`, écrivez une requête qui renvoie la liste des noms de tous les personnages du serveur.
- Dans `ex02.sql`, écrivez une requête qui renvoie les noms de tous les personnages morts du serveur (qui sont les personnages dont la santé est égale à zéro).
- Dans `ex03.sql`, écrivez une requête qui renvoie les noms des 5 premiers personnages, triés par expérience dans l'ordre décroissant.
- Dans `ex04.sql`, écrivez une requête qui renvoie le titre de toutes les quêtes, triées par la longueur de leur titre, dans l'ordre croissant.
- Dans `ex05.sql`, écrivez une requête qui renvoie les noms des personnages classés par ordre croissant d'avancement dans le jeu. Pour départager les personnages de même niveau, triez-les par expérience. Pour départager les personnages ayant la même expérience, triez-les en fonction de la quantité d'or qu'ils possèdent.

- Dans `ex06.sql`, écrivez une requête qui renvoie les titres des quêtes qui "valent le coup" pour votre personnage de niveau 10. Vous avez réalisé que cela vaut la peine de faire une quête lorsque la créature qui commence la quête est la même que celle qui la termine (parce que vous n'avez généralement pas à voyager beaucoup) **ou** que la récompense en or est supérieure à 100, et le niveau requis est d'au moins 8, mais pas plus de 10 (parce que vous ne pouvez pas faire ceux qui ont un niveau plus élevé).

Remarque : lorsque l'ordre des résultats n'est pas spécifié, vous pouvez renvoyer les résultats dans l'ordre que vous souhaitez. Cela s'applique à l'ensemble du sujet.

Niveau supérieur !

Inscrivez les réponses à cet exercice dans `ex07.sql`.

- Le joueur `LeKrogan` a gagné un concours lors d'un événement spécial. Ecrivez une requête qui lui donne le niveau 15.
- Le joueur `duck` s'est plaint que son niveau n'était pas mis à jour malgré le fait qu'il ait gagné suffisamment d'expérience. Écrivez une requête qui augmente son niveau d'une unité.
- Vous avez décidé de changer la façon dont la statistique `max_health` est calculée pour les personnages. Elle est maintenant de `(niveau + 1) * 10` si le personnage est un homme, et de `niveau * 10` si le personnage est une femme[5].

Ecrivez une requête qui met à jour cette statistique pour tous les personnages.

Créatures manquantes

Inscrivez les réponses à cet exercice dans `ex08.sql`.

Pour satisfaire les joueurs de haut niveau, vous avez décidé d'ajouter des créatures difficiles dans le jeu. Leur nom est 'Wild Yak", elles sont de niveau 99, leur ID de modèle 3D est 1387, elles donnent 1000 points d'expérience et 1000 pièces d'or lorsqu'elles sont tuées, leur santé est de 8, leur vitesse est de 8 et leur attaque est de 50.

Ajoutez trois instances de la créature "Young Wolf" (id = 1) pour qu'elle apparaisse aux positions :

- `(5, 6, 7)`
- `(-3, -2, -1)`
- `(42, 43, 44)`

Statistiques avancées

- Dans `ex09.sql`, écrivez une requête qui renvoie le nom de la créature dont l'identifiant est 6 (l'ID d'instance de la créature, et non de modèle).
- Dans `ex10.sql`, écrivez une requête qui retourne l'identifiant de toutes les créatures (les instances, pas les modèles) dont le niveau est strictement supérieur à 10.

- Dans `ex11.sql`, écrivez une requête qui renvoie tous les titres des quêtes et le nom des créatures qui les lancent.
- Dans `ex12.sql`, écrivez une requête qui renvoie le nom des créatures qui sont les mêmes au début et à la fin d'une quête.
- Dans `ex13.sql`, écrivez une requête qui renvoie le nom des créatures qui n'apparaissent jamais dans le jeu (le modèle existe mais n'est jamais "instancié").
- Dans `ex14.sql`, écrivez une requête qui renvoie le titre des quêtes que le personnage a accomplies.
- Dans `ex15.sql`, écrivez une requête qui renvoie la liste des noms des différentes quêtes et des personnages qui les ont commencées sans les terminer.

Faction

Inscrivez les réponses à cet exercice dans `ex16.sql`.

Pour comprendre quelles créatures sont amicales et lesquelles ne le sont pas, nous voulons avoir un moyen de savoir quelle est leur "faction".

Créez une table `faction` qui contient :

- Un champ `id`, clé primaire, type `SERIAL`.
- Un champ `color`, entier contenant la couleur de la faction.
- Un champ `name`, chaîne de 64 caractères maximum, contenant le nom de la faction.

Insérez dans votre table une faction avec l'id `0`, la couleur `0x00FF00` et le nom `Enlightened`.

Ensuite, ajoutez une colonne `faction_id` dans la table `creature`, qui est une clé étrangère à la table `faction`.

Suppression des utilisateurs inactifs

Pour éviter d'avoir une base de données trop grande, vous voulez supprimer tous les personnages inactifs.

Dans `ex17.sql`, écrivez une requête qui supprime tous les personnages qui n'ont pas été connectés depuis plus d'un an (31556926 secondes). Pour cet exercice, vous n'avez pas besoin de supprimer les lignes inutiles dans les autres tables.

Le 10 janvier 2016, un nombre important d'utilisateurs ont arrêté de jouer à votre jeu. Afin de déterminer les utilisateurs qui sont partis, vous voulez lister les personnages dont la dernière connexion remonte à cette date. Dans `ex18.sql`, écrivez une requête qui affiche le nom de tous les personnages dont la dernière connexion remonte à cette date.

Objets et inventaires !

Ajoutons des objets dans notre jeu !

Vous devrez créer 3 tables dans `ex19.sql` .

`item` contient les champs suivants :

- `id` , une clé primaire
- `name` , une chaîne de 64 caractères maximum, pour le nom de l'objet
- `rarity` , un nombre entier pour la rareté de l'article
- `icon_id` , un nombre entier qui contient l'ID de l'icône de l'article.

`inventory` est utilisé pour représenter une relation *many-to-many* entre les objets et les personnages. Une ligne signifie que le personnage spécifié possède l'objet spécifié. La table contient :

- `id` , une clé primaire
- `item_id` , une clé étrangère à l'ID de l'article
- `character_id` , une clé étrangère de l'ID du personnage.

`loots` est utilisée pour représenter une relation *many-to-many* entre les objets et les créatures. Une ligne signifie que la créature en question peut donner l'objet spécifié en cas de victoire. La table contient :

- `id` , une clé primaire
- `item_id` , une clé étrangère à l'ID de l'objet
- `creature_gid` , une clé étrangère de l'ID du modèle de créature.

BOSS FINAL

Dans `ex20.sql` , écrivez une requête qui renvoie la liste des noms des créatures que le personnage 'LeKrogan' a potentiellement tué pour obtenir les objets qu'il possède dans son inventaire.

BONUS (si vous avez fait tout le reste)

Créez une vue `v_character_stats` qui permet de faire un classement des personnages. Chaque `character` ne doit éxister **qu'une fois** au sein de cette vue. Elle doit afficher :

- le nom du personnage
- le niveau du personnage
- l'argent obtenu par le personnage
- le nombre de quêtes complétées
- le nombre de quêtes incomplètes
- le nombre d'objets dans son inventaire
- le nombre de monstres tués

Ce tableau de classement doit afficher les résultats ordonnés par :

- gold (de celui qui en a le plus vers le moins)
- quêtes complétées (de celui qui en a complétée le plus vers le moins)
- quêtes incomplètes (de celui qui en a le moins vers le plus)
- nombre d'objets dans l'inventaire (de celui qui en a le plus vers le moins)

1. http://en.wikipedia.org/wiki/Relational_algebra ↩️
2. Object Relational Mapping: http://en.wikipedia.org/wiki/Object-relational_mapping ↩️
3. http://en.wikipedia.org/wiki/Object-relational_impedance_mismatch ↩️
4. https://www.getmangos.eu/ ↩️
5. Pour assurer l'équité entre les femmes et les hommes, les personnages féminins disposent déjà d'un bonus en intelligence. ↩️