GOKTEKIN MUHAMMED EMIN

PROJET BASE DE DONNÉES PARTIE 2

Pour commencer, j'ai décidé de reprendre la correction fournis par le professeur, dans laquelle j'ai détecté quelques soucis. J'ai donc mis en commentaire deux lignes qui déclarent une clé étrangère sur les tables tache_en_cours et tache_fini. J'ai ensuite modifie l'attribut ref_score_categorie_tache présent dans les tables comporte et score_categorie_tache en un entier car dans la partie PL/SQL cela a été nécessaire. Ce sont les seules modifications que j'ai apportées au fichier Tables.SQL.

Partie requête pl/sql:

1) Pour la première requête j'ai inséré des données spécifiques pour tester la validité de la requête.
   J'ai choisie de créer deux index sur liste de tâche et sur utilisateur pour faciliter la recherche sur le pays.
2) Dans cette requête j'ai créé trois requêtes sur les jointures pour pouvoir améliorer l'accès aux données, le tri et la sélection des lignes dans les tables concernés.
3) Les index créés permettent une amélioration à l'accès pour les différents utilisateurs à leur tâche comprise dans les tables de tâches.
4) Pour cette requête les index ont été mis en place pour faciliter l'accès aux données lors de la recherche des correspondances entre Tâche et Dépend_de.
5) Dans cette requête les index que j'ai mis en place sont aussi sur la jointure pour pouvoir accéder plus rapidement aux données et faciliter le tri.

Procédures et fonctions SQL:

1) Pour la fonction CalculerScoreUtilisateur qui prend en argument un utilisateur et qui nous donne le score d'un utilisateur, j'ai utilisé la fonction COALESCE pour le cas où le score est nul d'afficher 0. Le calcul du score des tache est effectué en utilisant la table score_categorie_tache qui associe a chaque tâche un score (on additionne le score de chaque tâche). Pour les tâches non effectuées un malus est donné (-2). J'ai mis du code pour pouvoir exécuter la procédure avec différents utilisateurs.

2) Pour la fonction archiver_taches_passees, j’ai décidé d’utiliser le statut d’une tâche. C'est-à-dire que les tâches terminées avec une date de réalisation antérieure à une semaine à partir du début de la semaine actuelle sont mises à jour avec le statut 'Archivée'. Et les tâches en cours avec une date d'échéance antérieure à une semaine à partir du début de la semaine actuelle et qui sont encore en cours sont mises à jour avec le statut 'Archivée'.

3) Pour cette dernière fonction TrouverUtilisateursSimiliaires j’ai choisi d’utiliser deux curseurs:

   - Curseur cur_utilisateurs_similaires, ce curseur sélectionne les utilisateurs similaires à l'utilisateur de référence en se basant sur le nombre minimal de tâches similaires, l’utilisation de la table est_assigne permet de déterminer les tâches assignées.

   - Curseur cur_taches_similaires, ce curseur sélectionne les tâches similaires entre l'utilisateur de référence et l'utilisateur similaire. La condition dans la clause EXISTS garantit qu'il y a un nombre minimal de mots en commun dans les tâches.

   La boucle principale parcourt les utilisateurs similaires et pour chacun d'eux, elle parcourt les tâches similaires et tout cela est affiché à l’aide de DBMS_OUTPUT.PUT_LINE.

   En cas d’erreur une exception est envoyée.

Contraintes d’intégrités et triggers:

Pour le score associés à différentes tâches j’ai du ajouter une contraintes de clé étrangères sur la table Score_categorie_tache la clé est la suivante:

ALTER TABLE Score_categorie_tache ADD CONSTRAINT FK_SCORE_REFTACHE FOREIGN KEY (ref_score_categorie_tache) REFERENCES Tache (ref_tache);

Hormis cette clé il y a différentes clés étrangères sur les tables comporte, contient, depend_de , est_assigne, liste_tache, tache_fini, tache_en_cours, travaille. Qui permettent la relation entre les tâches et les utilisateurs et leurs différentes nécessités.

Pour les triggers:

1) J'ai créé un déclencheur nommé maj_score_utilisateur permettant de mettre à jour le score en fonction du statut de la tâche. Pour cela j'ai utilisé le "statut" d'une tâche de la table tache_fini et l'attribut "score" de la table utilisateur qui sera modifié en fonction du résultat attendu et du statut de la tâche. Si le statut est modifié en "terminée" alors le score est augmenté avec le bon nombre, si il est modifié en "archivée" le score est diminué de 2.

2) Pour le déclencheur nommé trig_creation_taches_periodiques, on commence par vérifier si la périodicité spécifiée a une date de fin, ensuite la référence de l'utilisateur est récupérée à partir de la table tache_en_cours en utilisant la référence de tâche périodique.

   Une boucle est utilisée pour créer les tâches associées en fonction de la périodicité qui sont compris entre les dates de début et fin, dans laquelle on va mettre à jour ou insérer une tâche dans la table tache_en_cours en fonction de la date actuelle. La déclaration "merge into" est utilisée pour fusionner les données dans la table Tache_en_cours en fonction de la référence de tâche périodique. Si une correspondance est trouvée, la tâche existante est mise à jour. Si aucune correspondance n'est trouvée, une nouvelle tâche est insérée.

Pour ces déclencheurs, le test est contenu sous la déclaration.

Toutes les parties codes se trouvent dans la partie script.sql et les insertions se trouvent dans la partie insert.sql (il y a une partie pour la requête 1 et une pour la fonction 1 pl/sql). Il se peut que certaines insertions de tache_en_cours et tache_fini ne marchent pas, il faut juste essayer une deuxième fois et normalement ce sera bon.

Les index sont présents avec chacune des requêtes et pour chaque fonction nécessitant un affichage j'ai mis un script permettant l'affichage sous la fonction dans le fichier script.sql.