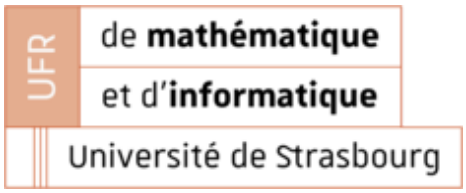

L3 Informatique

Automne 2023

Projet Bases de Données 2
Partie 2

Justine Andreolli

Seyedali Hashemi

Groupe TD2 TP3

Table des matières

1 Introduction

Nous avons décidé de reprendre les fichiers fournis pour la correction de la Partie 1 en prenant la liberté de modifier certaines variables et de préciser certaines foreign key. Voici la structure de nos fichiers sql :

create.sql : script qui permet de lancer la création des tables, l'insertion de donnée, de tester les procédures/fonction/déclencheurs

data.sql : insertion de données dans les tables

declencheurs.sql : les deux déclencheurs demandés

dropTables.sql : suppression des tables et des contraintes

index.sql : index nécessaires pour la partie sur les requêtes

procedures_fonctions.sql : les trois procédures/fonctions demandées

requetes.sql : les cinq requêtes demandées

tableCreate.sql : script de création de table

triggers.sql : les triggers que nous avons décidé d'implémenter et ceux demandés pour la partie 1

tests.sql : les appels pour tester les procédures/fonctions/déclencheurs

2 Modèle Entité-Association (EA)

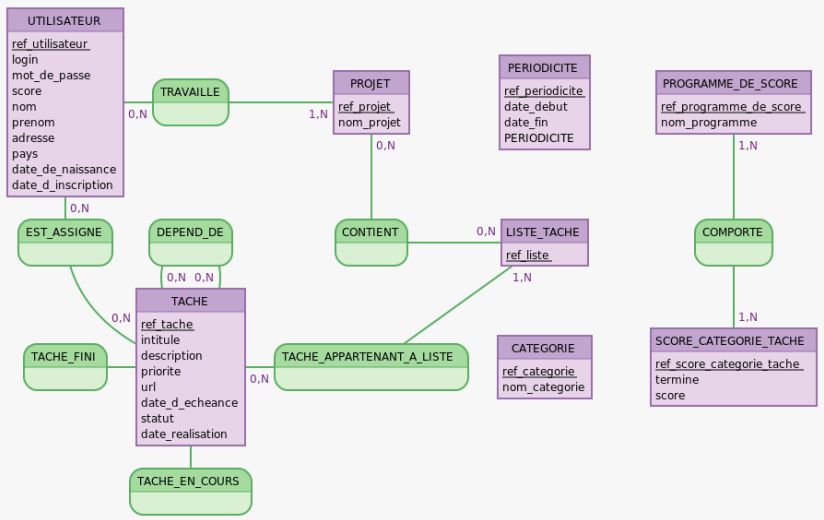

Figure 1: Modèle Entité-Association

Nous avons également mis l'image dans notre zip pour une meilleure lisibilité.

3 Système de score

Pour le système de score, nous avons décidé d'attribuer des points à valeur positives si la tâche a été terminé avant la date d'échéance et des points à valeurs négatives si la tâche n'a pas été réalisé avant la date d'échéance ou si elle n'a pas été réalisée du tout.

Pour calculer le score, nous trions d'abord les tâches par date de réalisation ou à défaut la date d'échéance puis nous ajoutons ou retirons les points.

Puisque nous avons des points négatifs, le tri par ordre de réalisation/date d'échéance de la tâche est très important car si la valeur du score de l'utilisateur est négative, nous la ramenons à zéro. C'est pour cela en partie que nous avons du faire des choix pour la partie 5 sur les déclencheurs demandés.

4 Choix des index

Pour la première requête nous avons décidé de créer l'index *idx_utilisateur_ref_nom_prenom_pays*.

Pour la troisième requête nous avons décidé de créer l'index *idx_usr* qui regroupe les variables login, nom, prenom et adresse qui sont demandés.

Pour la cinquième requête nous avons décidé de créer l'index *idx_tache_fini_utilisateur_categorie_realisation* et également l'index *idx_score_categorie_nom_score*.

5 Partie sur les déclencheurs

Pour le **premier déclencheur**, comme nous l'avons expliqué dans la partie 3 sur la décision du score, notre implémentation fait que si l'on calcule le score de l'utilisateur dès qu'une tâche est terminée avec un trigger, si l'ordre dans lequel les tâches ont été ajoutées ne sont pas ascendantes en fonction de la date de réalisation ou le cas échant de la date d'échéance. Alors le score sera faux.

Pour nous prévenir de cela, nous avons décidé de créer une procédure plutôt qu'un trigger qui trie dans le bon ordre les tâches par utilisateur avant de calculer le score. Toutefois, l'algorithme reste similaire à si nous avions fait un trigger.

Pour le **deuxième déclencheur**, nous avons décidé de l'implémenter sous forme de procédure. Cette procédure a pour fonction de dupliquer les tâches périodiques et de les enregistrer dans la table Tache. Nous avons opté pour une procédure plutôt qu'un déclencheur car, dans notre cas et en particulier avec Oracle, l'utilisation d'un déclencheur provoquerait une erreur de table mutante,

car il est techniquement inadéquat qu'un déclencheur effectue simultanément des opérations de lecture et d'écriture sur la même table (Tache). La procédure fonctionne sans problèmes et assure la duplication des tâches périodiques jusqu'à la date de fin spécifiée

6 Choix sur les requêtes

Pour les requêtes 2 et 5 sur le score, nous avons décidé de d'abord trier les requêtes en fonction de leur date de réalisation ou à défaut leur date d'échéance pour ne pas interférer avec la logique du score que nous avons décidé d'implémenter.

Pour la quatrième requête qui affiche les dépendances, nous avons décidé de calculer toutes les dépendances d'une tâche. C'est à dire que si une tâche A est dépendante d'une tâche B mais que celle-ci est également dépendante d'une tâche C. Alors la tâche A est dépendante de deux tâches pas seulement d'une.

7 Choix des triggers

Nous avons repris les triggers sur le mot de passe et sur le login demandés pour la partie 1.

Nous avons créé le trigger *assign_task_trigger* qui créé les données nécessaires pour lier un utilisateur à une tâche dans la table EST_ASSIGNE lorsqu'une tâche est créée.

Nous avons créé le trigger *en_cours_task_trigger* qui duplique une tâche avec le statut en cours dans la table tache_en_cours.

Nous avons mis en place un trigger *check_category_trigger* qui vérifie que les catégories de tâches appartenant à la même liste sont les mêmes.

Finalement nous avons créé plusieurs trigger : *CheckDatesInscription*, *CheckTaskDates*, *CheckDependencyDates*, *CheckDatePeriodicite* et *CheckTaskCompletionDate* qui vérifient que les dates entrées sont en accord avec les autres dates pour ne pas avoir de données incohérentes.

8 Procédures et Fonctions PL/SQL

La **procédure 1** nous avons repris en partie le code pour calculer le score de la semaine de la requête 5 car l'énoncé était similaire.

Pour la **procédure 2** qui archive les tâches, celles avec le statut 'fini' sont automatiquement déplacées chaque semaine vers la table *Tache_fini*. Cette opération est supposée être automatisée, car nous n'avons pas la capacité de lancer des jobs directement sur Oracle.

Pour la **procédure trois** qui suggère des tâches à un utilisateur, nous avons

décidé de créer plusieurs fonctions. La dernière fonction *Procedure3AfficheTachesSuggerees* est celle qu'il suffit d'appeler pour produire le résultat car c'est elle qui appelle tous les autres fonctions mais nous avons également fourni dans *tests.sql* des tests sur les autres fonctions. La fonction *are_tasks_similar* permet de vérifier si deux tâches sont similaires, *are_users_similar* vérifie si deux utilisateurs sont similaires en appelant la fonction précédente et finalement *Procedure3AfficheTachesSuggerees* appelle la fonction précédente et produit le résultat souhaité.

9 Les définitions de contraintes d'intégrité

Nous présentons ici les **Primary Key**, **Foreign Key**, **Unique Constraint** et **Check Constraint** de chaque table.

9.1 Table Projet

PRIMARY KEY (ref_projet) → garantit l'unicité

UNIQUE(nom_projet) → nous avons fait le choix de garder l'implémentation de la correction qui lie certaines tables avec *nom_programme* plutôt que *ref_programme* donc il fallait rajouter cette contrainte.

9.2 Table Categorie

PRIMARY KEY (ref_categorie) → garantit l'unicité

UNIQUE(nom_categorie)

9.3 Table Comporte

PRIMARY KEY (nom_programme, ref_score_categorie_tache) → garantit l'unicité

→ assure que la variable existe bien dans la table

FOREIGN KEY (ref_score_categorie_tache) → assure que la variable existe bien dans la table Score_categorie_tache

FOREIGN KEY (nom_programme) → assure que la variable existe bien dans la table Programme_de_score

9.4 Table Contient

PRIMARY KEY (nom_projet, ref_liste) → garantit l'unicité

FOREIGN KEY (ref_liste) → assure que la variable existe bien dans la table Liste_tache

FOREIGN KEY (nom_projet) → assure que la variable existe bien dans la table Projet

9.5 Table Depend_de

PRIMARY KEY (ref_tache_1, ref_tache_2) → garantit l'unicité

FOREIGN KEY (ref_tache_2) → assure que la variable existe bien dans la table Tache

FOREIGN KEY (ref_tache_1) → assure que la variable existe bien dans la table Tache

9.6 Table Est_assigne

PRIMARY KEY (ref_utilisateur, ref_tache) → garantit l'unicité

FOREIGN KEY (ref_tache) → assure que la variable existe bien dans la table Tache

FOREIGN KEY (ref_utilisateur) → assure que la variable existe bien dans la table Utilisateur

9.7 Table Liste_tache

PRIMARY KEY (ref_liste) → garantit l'unicité

FOREIGN KEY (ref_utilisateur) → assure que la variable existe bien dans la table Utilisateur

FOREIGN KEY (nom_categorie) → assure que la variable existe bien dans la table Categorie

9.8 Table Periodicite

PRIMARY KEY (ref_periodicite) → garantit l'unicité

9.9 Table Score_categorie_tache

PRIMARY KEY (ref_score_categorie_tache) → garantit l'unicité

9.10 Table Tache

PRIMARY KEY (ref_tache) → garantit l'unicité

FOREIGN KEY (ref_utilisateur) → assure que la variable existe bien dans la table Utilisateur

FOREIGN KEY (ref_periodicite) → assure que la variable existe bien dans la table Periodicite

FOREIGN KEY (nom_categorie) → assure que la variable existe bien dans la table Categorie

CHECK (priorite = 'L' OR priorite = 'M' OR priorite = 'H') → L pour low, M pour medium et H pour high

CHECK (statut IN ('en cours', 'fini') OR statut IS NULL) → le statut peut uniquement prendre trois valeurs. Fini signifie que la tâche a été réalisée.

FOREIGN KEY (ref_tache) → assure que la variable existe bien dans la table Tache

9.11 Table Tache_en_cours

PRIMARY KEY (ref_tache) → garantit l'unicité

FOREIGN KEY (ref_utilisateur) → assure que la variable existe bien dans la table Utilisateur

FOREIGN KEY (ref_periodicite) → assure que la variable existe bien dans la table Periodicite

FOREIGN KEY (nom_categorie) → assure que la variable existe bien dans la table Categorie

CHECK (priorite = 'L' OR priorite = 'M' OR priorite = 'H') → L pour low, M pour medium et H pour high

CHECK (statut IN ('en cours', 'fini') OR statut IS NULL) → le statut peut uniquement prendre trois valeurs. Fini signifie que la tâche a été réalisée.

FOREIGN KEY (ref_tache) → assure que la variable existe bien dans la table Tache

9.12 Table Tache_fini

PRIMARY KEY (ref_tache) → garantit l'unicité

FOREIGN KEY (ref_utilisateur) → assure que la variable existe bien dans la table Utilisateur

FOREIGN KEY (ref_periodicite) → assure que la variable existe bien dans la table Periodicite

FOREIGN KEY (nom_categorie) → assure que la variable existe bien dans la table Categorie

CHECK (priorite = 'L' OR priorite = 'M' OR priorite = 'H') → L pour low, M pour medium et H pour high

CHECK (statut IN ('en cours', 'fini') OR statut IS NULL) → le statut peut uniquement prendre trois valeurs. Fini signifie que la tâche a été réalisée.

FOREIGN KEY (ref_tache) → assure que la variable existe bien dans la

table Tache

9.13 Table Tache_appartenant_a_liste

PRIMARY KEY (ref_liste, ref_tache) → garantit l'unicité

FOREIGN KEY (ref_tache) → assure que la variable existe bien dans la table Tache

FOREIGN KEY (ref_liste) → assure que la variable existe bien dans la table Liste_tache

9.14 Table Travaille

PRIMARY KEY (nom_projet, ref_utilisateur) → garantit l'unicité

FOREIGN KEY (ref_utilisateur) → assure que la variable existe bien dans la table Utilisateur

FOREIGN KEY (nom_projet) → assure que la variable existe bien dans la table Projet

9.15 Table Programme_de_score

PRIMARY KEY (ref_programme_de_score) → garantit l'unicité

UNIQUE(nom_programme

FOREIGN KEY (nom_categorie) → assure que la variable existe bien dans la table Categorie

CHECK (termine = 'Y' or termine = 'N') → Y signifie que la tâche est réalisée avant la date d'échéance et N que la date n'a pas été réalisée avant la date d'échéance ou pas du tout.

9.16 Table Utilisateur

PRIMARY KEY (ref_utilisateur) → garantit l'unicité

UNIQUE(login)

FOREIGN KEY (nom_programme) → assure que la variable existe bien dans la table Programme_de_score

10 Modèle Entity Relationship Diagram (ERD)

Nous avons également mis l'image dans notre zip pour une meilleure lisibilité.

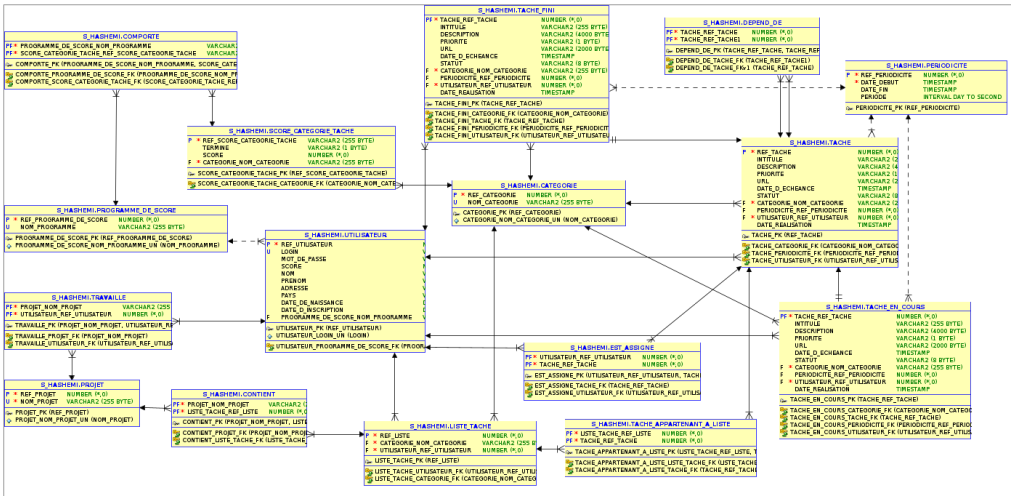

Figure 2: Modèle Entity Relationship Diagram depuis sqldeveloper