Rapport de projet – Base de données

Base de données pour une application de gestion de planning

Auteurs : Guillaume Dumas
Jérémy Grelaud

Professeur : Karam Elie Bou Chaaya

Sommaire

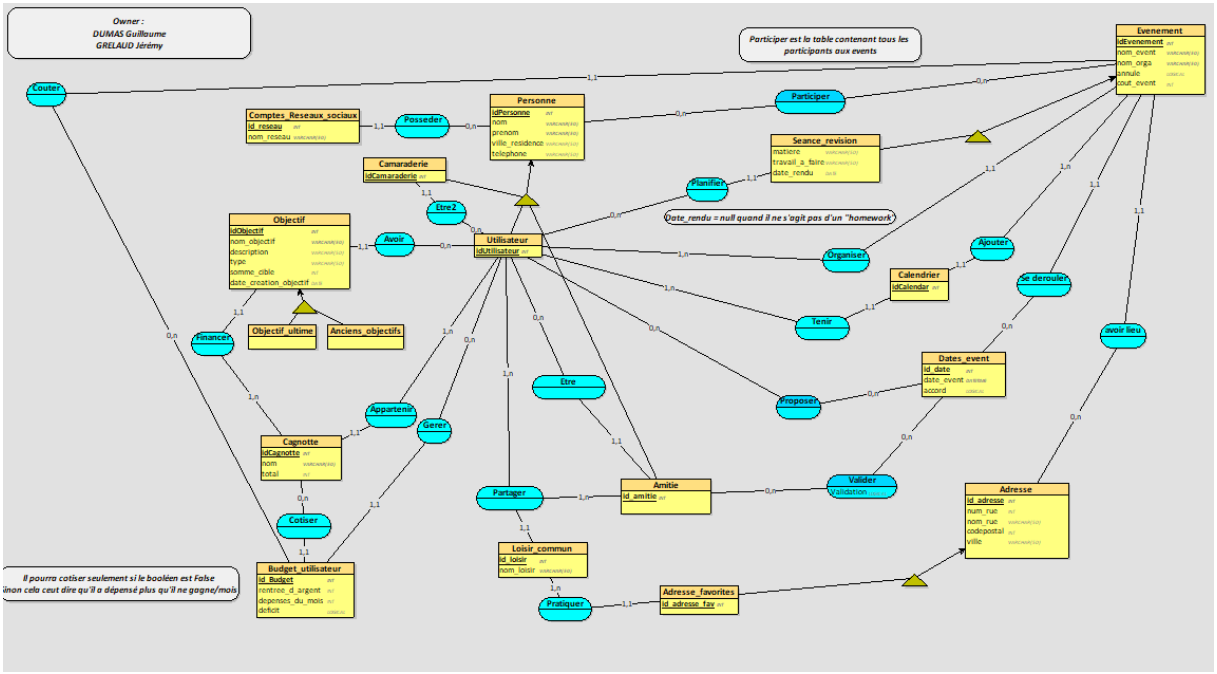
1 Introduction

Le but de ce projet est de créer la base de données d'une application de planification, puis de l'exploiter et extraire certaines données spécifiques à l'aide de requêtes et de vues. Nous allons remplir cette base de données avec un certain nombre de valeurs, que nous jugerons représentatives d'une base complète. Cela nous permettra de tester le bon fonctionnement de nos requêtes et de nos vues.

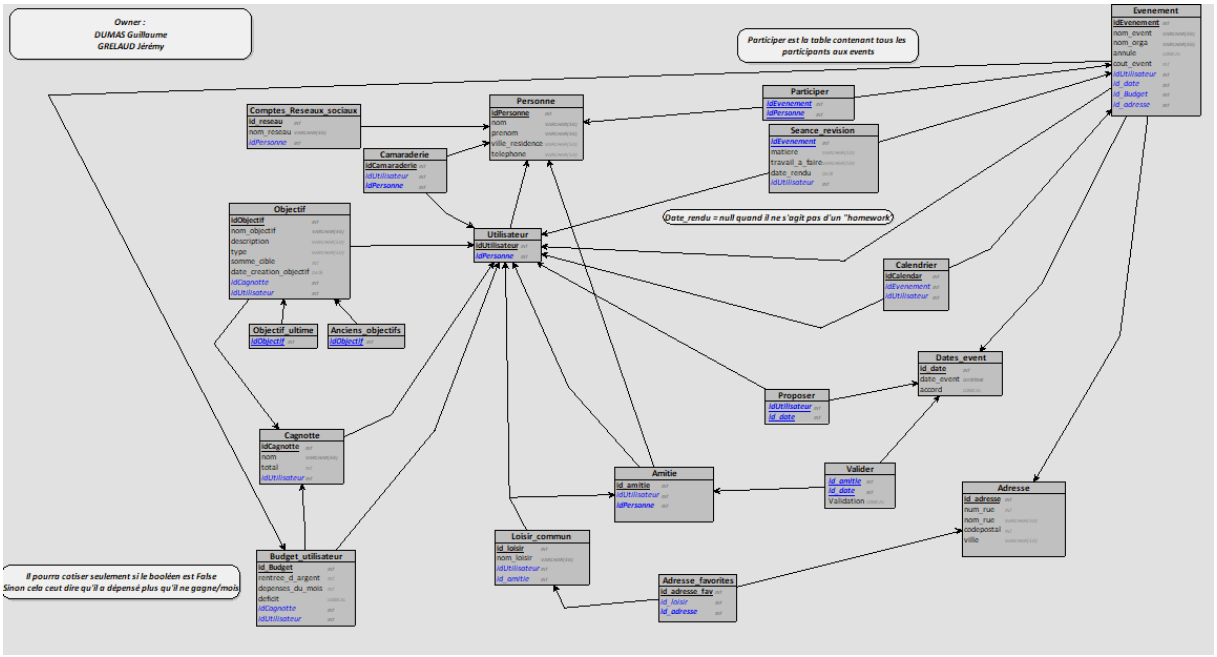
2 Création de la base de données

2.1 MCD et MLD

MCD



MLD



2.2 Description de la base de données

Nous avons considéré notre base de données comme une application regroupant plusieurs utilisateurs, ici 10. Il existe donc plusieurs cagnottes, calendriers, liste d'objectifs ou encore de camarades de classe. De plus, certaines données, relatives aux sommes et dépenses d'argents dans les budgets ou cagnottes des utilisateurs sont rentrées en « dur » dans la base de données et considérées mise à jour en temps réel. Il est important de noter que les tables « *Amitie* » et « *Camaraderie* » sont représentées comme des relations et non une liste de personne, il y a donc un tuple de 2 personnes dans chaque ligne qui sont soit en relation d'amitié soit camarades de classe.

3 Remplissage de la base avec un jeu de données

Pour le jeu de données, on va créer plusieurs entrées manuellement. Nous allons notamment créer 10 personnes, les associer selon des amitiés et des camaraderies. Nous allons créer également une douzaine de dates, de cagnottes, d'objectifs, de loisirs d'adresses et d'évènements. Les personnes utilisant notre application sont considérées comme des « étudiants » d'une même école n'étant pas forcément amis ou dans la même classe mais pouvant interagir entre eux.

4 Requêtes

Cette partie présente une rapide explication de chaque requête.

Requête 1 :
On sélectionne les éléments de l'évènement où la date associée correspond au maximum de toutes les dates, tout en étant plus petites que la date d'aujourd'hui (c'est-à-dire que nous ne considérons que les dates antérieures qui ont déjà eu lieu).

Requête 2 :
On sélectionne d'abord, avec une sous-requête, tous les identifiants des amis liés à un utilisateur (donné en paramètre) qui ont un loisir spécifique (donné en paramètre) dans la table « *loisir_commum* ». Ensuite, on sélectionne les informations des amis qui sont dans la liste de la sous- requête.

Requête 3 :
On compte le nombre de ligne dans la table « *seance_revision* » où le mois de la date de la séance de révision vaut le mois d'aujourd'hui moins 1 (donc le mois dernier). On utilise la fonction *now()* pour avoir la date d'aujourd'hui. On affiche ensuite le résultat de cette somme. La deuxième requête dans le script permet d'afficher toutes les infos relatives aux séances de révisions du mois dernier.

Requête 4 :
Dans une sous-requête, on sélectionne les personnes amis de l'utilisateur donné en paramètre. Puis on sélectionne les camarades du même utilisateur qui ne sont pas (« *NOT IN* ») dans la liste précédente obtenue grâce à la sous-requête.

Requête 5 :
On sélectionne la ville liée à un événement donné en paramètre, puis on sélectionne parmi les amis d'un utilisateur donné en paramètre ceux qui habitent dans une ville contenue dans la liste de la sous-requête. Le fait de comparer la ville d'un ami avec une liste (en utilisant « *in* ») permet de gérer le cas où un évènement se déroule dans plusieurs villes.

Requête 6 :
Pour chaque événement non annulé et dont la date est antérieure à aujourd'hui, on sélectionne les identifiants des adresses d'événements. Puis on sélectionne les adresses favorites qui ne sont pas dans la liste de la sous-requête. Ainsi, on obtient les adresses favorites où il ne s'est jamais tenu d'événements jusqu'à maintenant.

Requête 7 :
Les dépenses et les rentrées d'argent du mois de l'utilisateur sont rentrées en « dur » dans le budget qui lui est associé. Ces données sont censées être mises à jour chaque mois. La requête consiste alors simplement à afficher la colonne « *rentree_d_argent* » et « *depenses_du_mois* ». Il est aussi possible dans notre base de données d'afficher les coûts d'organisations d'événements du mois actuel mais cette somme est considérée comme déjà comprise dans la colonne dépense du mois du budget utilisateur.

Requête 8 :
On compte le nombre d'événements maintenus et organisés par l'utilisateur donné en paramètre (ici, l'utilisateur 2). On divise ensuite cette somme par le nombre d'événements organisés par l'utilisateur donné en paramètre. Puis on multiplie le tout par 100 pour obtenir le taux d'acceptation sous la forme d'un pourcentage.

Si l'utilisateur en question n'a organisé aucun événement, le taux d'acceptation sera « *null* ».

5 Vues

Cette partie présente une rapide explication de chaque vue.

Vue 1 :
On affiche les objectifs de l'utilisateur choisi datant de l'année précédente. Pour ce faire, on joint les deux tables « *objectif* » et « *cagnotte* » et on sélectionne les objectifs dont la date de création vaut l'année actuelle moins 1.

Vue 2 :
On compte le nombre d'événements organisé par chaque organisateur dans la table « *événements* » puis on se restreint aux personnes de la liste d'amis de l'utilisateur donné en paramètre. Enfin, on trie par ordre décroissant en fonction du nombre d'événements organisés par chacun, en se limitant aux deux premiers (avec « *limit 0,2* »). Comme le tri est décroissant, les deux premiers correspondent aux deux tops organisateurs parmi les amis.

Vue 3 :
On sélectionne chaque événement présent dans le calendrier de l'utilisateur sélectionné et dont la date correspond à celle d'aujourd'hui. On trie ensuite les dates par ordre croissant.

Vue 4 :
On regarde tous les amis de l'utilisateur sélectionné puis on compte le nombre de dates refusées dans la table « *valider* » par chacun de ces amis. On trie ensuite dans l'ordre décroissant du nombre de dates refusées et on se limite au premier élément, ce qui nous donne le trouble-fête, l'ami ayant refusé le plus de dates.

Vue 5 :
On regarde tous les amis de l'utilisateur sélectionné puis on compte pour chaque réseau social le nombre d'amis l'utilisant. On ordonne le tout dans l'ordre décroissant en se limitant au premier, ce qui nous donne le réseau social où sont actifs le plus grand nombre d'amis.

Vue 6 :
Pour chaque objectif de l'utilisateur sélectionné, on regarde si la somme cible est quatre fois supérieure à toutes les cagnottes annuelles de cet utilisateur. Si cela est bien le cas, on affiche l'objectif dans la liste.

Vue 7 :
On regarde dans la table « *valider* » les horaires annulés grâce à la fonction « *time()* », permettant de se limiter à la partie horaire de la date. On compte les occurrences de chaque horaire refusé et on trie le tout dans l'ordre décroissant en se limitant au premier élément, ce qui nous donne l'horaire le plus refusé dans notre application.

Vue 8 :
On compte le nombre de fois où nous avons révisé avec des camarades de classes avec lesquels nous ne sommes pas amis. On se limite aux deux premiers en triant les occurrences dans l'ordre décroissant. Il faut exclure l'utilisateur choisi, car il peut réviser avec lui-même mais ne deviendra pas amis avec lui-même. Il faut aussi se limiter aux séances de révisions organisées par l'utilisateur choisi OU aux séances de révisions auxquelles l'utilisateur choisi a participé.

6 Conclusion

Ce projet nous aura permis de mettre en application toutes nos nouvelles connaissances liées aux bases de données et à leur manipulation. Nous avons ainsi pu perfectionner nos connaissances du langage SQL.

Il nous aura également permis de découvrir de nouvelles choses. Nous avons pu expérimenter la difficulté de construire une bonne base de données, sous peine de nous retrouver bloqués lors de l'établissement des requêtes et des vues SQL.

Nous restons donc dans l'attente d'un autre projet tout aussi formateur.