Année universitaire : 2021/2022

Projet : S1.04

Création d’une base de données

Yifru Abenezer

Rouillon Tom

Welty Alexandre

Table des matières

I. Introduction

Avec cette application, on doit avoir la possibilité de créer un utilisateur, caractérisé par une adresse mail unique, et s'il possède une voiture, il doit pouvoir en informer l'application et remplir toutes les informations nécessaires.

Les utilisateurs possédant une voiture doivent pouvoir offrir des trajets, et tous les utilisateurs doivent pouvoir réserver un trajet.

De plus, les utilisateurs doivent pouvoir s'ajouter en amis, avec autant d'utilisateurs qu'ils veulent. Ces amis peuvent créer ensemble un groupe permettant à un de ses membres de proposer un trajet privé, que seuls les membres du groupe peuvent recevoir.

Ensuite, quand un utilisateur réserve un trajet, l'offreur doit recevoir une notification lui permettant d'accepter ou non, et celui qui réserve doit pouvoir recevoir cette réponse.

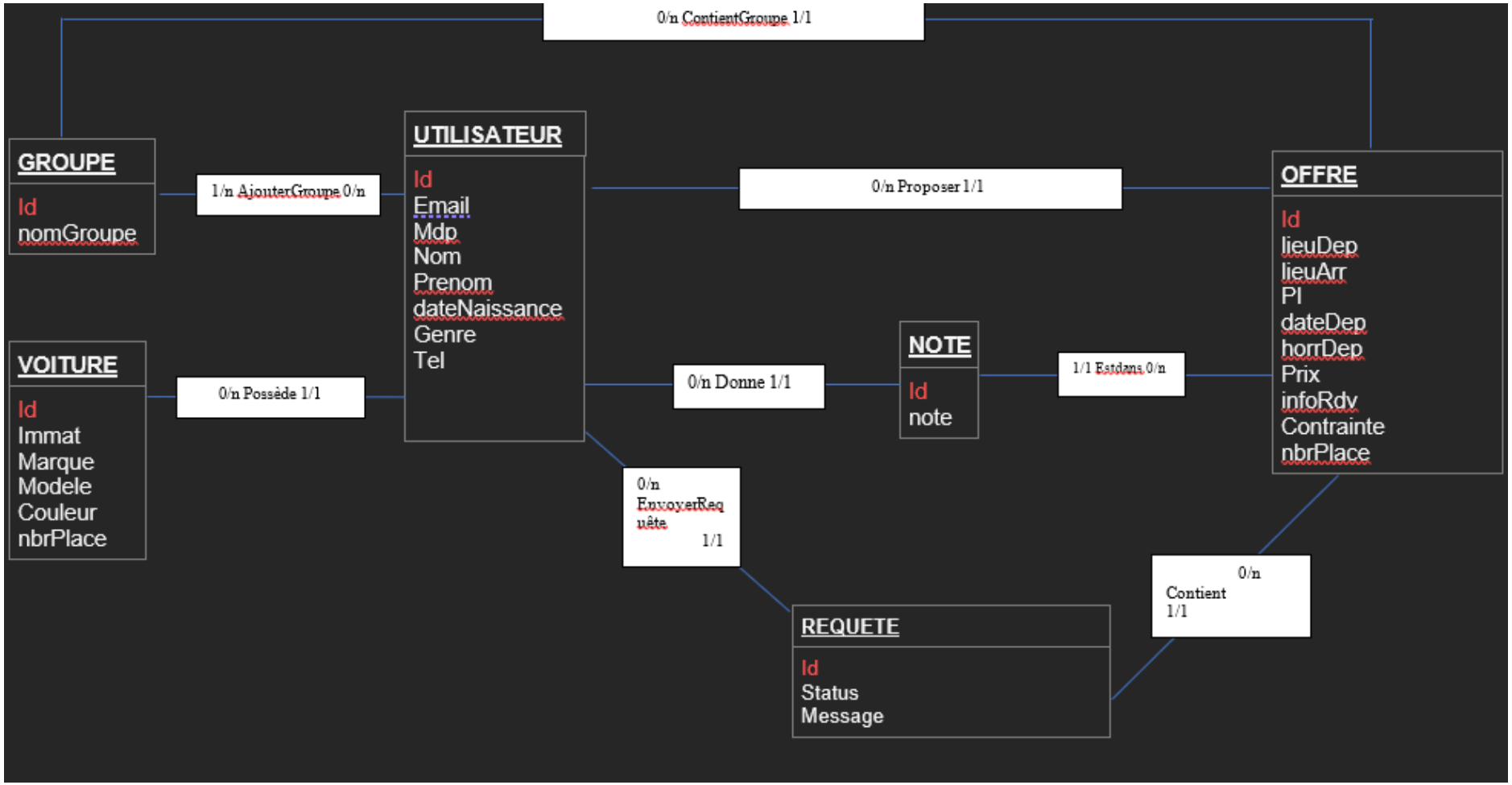
Quand l'offre est mise dans un groupe d'amis, tous les amis doivent recevoir une notification leur disant que l'offre est réservée, et donc n'est plus disponible.

Enfin, à chaque trajet que chaque utilisateur fait, que ce soit en tant que passager ou conducteur, ils peuvent mutuellement attribuer une note à l'autre.

II. Tableau des entités, attributs, associations et cardinalités

Entité	Attribut	Association
utilisateur	**id**, email, mdp, nom, prenom, dateNaissance, genre, tel,voiture(bool),immatriculation, marque, modele, couleur, nombreDePlace	Utilisateur 1–n groupe Utilisateur 1–n offre Utilisateur 1–n note
voiture	**id**, immatriculation, marque, modele, couleur, nombreDePlace	Voiture 1–1 Utilisateur
offre	**idTrajet**, lieuDep, lieuArr, PI, dateDep, horrDep, prix, infoRdv, contrainte, nbrPlace	Offre n-1 groupe
requête	**idReq**, status(oui/non), message	Requête n-1 offre
groupe	**idGroupe,** nomGroupe	Requête n-1 utilisateur
note	**idNote**, note	Note 1-1 offre

III. Modèle Entités/Associations



IV. Modèle Relationnel

Utilisateur(**id**,email,mdp,nom,prénom,dateNais,genre,tel)

Voiture(**id**,immat,marque,modèle,couleur,nbrPlace, *id_utilisateur(propriétaire)*)

Offre(**Id**,lieuDep,lieuArr,PI,dateDep,horrDep,prix,infoRDV,contrainte,nbrPlace,*Id_groupe,Id_membre*)

Note(**Id**,note,*id_utilisateur(offreur),id_utilisateur(demandeur),id_offre*)

Groupe(**Id**,nomGroupe)

Requête(**Id**,status,message,*id_membre(offreur),id_membre(demandeur),id_offre*)

UtilsiateurEstDansGroupe(*id_utilisateur,id_groupe*)

Utilisateur :

Si un utilisateur possède une voiture, alors le champ siVoiture prend la valeur 0, sinon il prend la valeur 1. Si c'est 0, alors l'utilisateur devra remplir les informations suivantes : l'immatriculation, la marque, le modèle, la couleur et le nombre de place. Ainsi, si un utilisateur veut proposer une offre de trajet, alors on peut imaginer que l'application pourra regarder dans la base de données le champ siVoiture pour un utilisateur donné : s'il est égal à 0, alors c'est bon, sinon l'application lui dira qu'il doit remplir les informations sur sa voiture. Ensuite, on peut imaginer que le champ siVoiture sera mis à jour à 0. Il n'y aura donc plus de problème si ce même utilisateur veut proposer une autre offre de trajet.

Groupe :

Un utilisateur peut être dans plusieurs groupes d'amis. Chaque groupe possède un id permettant de différencier tous les groupes, même si un utilisateur est dans plusieurs d'entre d'eux. Le nom du groupe permet d'éviter les redondances : en effet, si un utilisateur veut créer plusieurs groupes avec le même nom, l'application pourra rechercher dans la base de données et lui afficher un message d'erreur s'il possède déjà un groupe avec le même nom.

Offre :

Un utilisateur, s'il possède une voiture, peut proposer autant d'offre qu'il veut. A la fin, il peut choisir avec le champ groupe, s'il veut que cette offre soit publique ou qu'elle soit privée. Si elle est privée, alors seuls les utilisateurs appartenant au groupe pourront voir l'annonce.

Note :

Un utilisateur peut mettre autant de note qu'il veut, avec cependant un maximum d'une note par trajet. L'id_utilisateur_offre correspond donc à l'utilisateur qui donne la note, et l'id_utilisateur correspond à l'utilisateur qui reçoit la note. L'idOffre permet donc de faire en sorte qu'une seule note puisse être donnée par trajet et par utilisateur.

De plus, pour avoir la moyenne des notes, on pourrait avoir 2 requêtes du style :

SELECT COUNT(*) FROM Note, Utilisateur WHERE Note.id_utilisateur = Utilisateur.email (permet d'avoir le nombre de trajet noté pour un utilisateur)

Et

SELECT SUM(Note) From Note, Utilisateur WHERE Note.id_utilisateur = Utilisateur.email (permet de récupérer la somme de toutes les notes données pour un utilisateur)

Ainsi la division du résultat de la deuxième requête par la première nous permet d'avoir la moyenne des notes pour un utilisateur.

Requête :

Un utilisateur, en tant que passager, peut envoyer autant de requête qu'il veut, càd réserver autant de trajet qu'il veut. Ainsi, l'utilisateur voulant réserver un trajet peut envoyer une requête, grâce à l'écran de recherche, à l'utilisateur proposant le trajet. Celui-ci peut ainsi donner une réponse favorable ou non (0 ou 1 dans le champ status), et une nouvelle notification sera envoyée à l'utilisateur voulant réserver, contenant la réponse. On peut imaginer que l'application regardera le champ status et tant qu'il sera null (donc quand la requête est envoyée), elle ne fera rien, et ce sera uniquement quand il y aura une valeur qu'elle envera une notification.

De plus, si un groupe est associé à l'offre, alors le système de notification est le même, à l'exception près qu'une notification est envoyée à tous les membres du groupe quand la réservation est terminée (quand le nombre de place est atteint ou quand l'offrant le décide).

Recherche :

Pour la recherche d'un trajet, on peut imaginer que l'application, avec les informations que l'utilisateur rentre (càd la ville de départ, d'arrivée et la date), pourra, avec une requête SQL, prendre à partir de la table Offre les trajets, avec les prénoms des offreurs, correspondant à ce que l'utilisateur recherche.

Ainsi, il pourra envoyer une requête, avec un message ou non, à l'offreur qui pourra lui répondre.

I. Contraintes d'intégrités

Table Utilisateur	Table Offre	Table Note	Table Requête	Table Groupe
- **Id** : clé primaire, auto-incrementation, non nul - Email : non nul, unique - Mdp : non nul - Nom : non nul - Prénom : non nul - DateNaissance : non nul - Genre : non nul - Tel : non nul - siVoiture : nul - Immatriculation ; nul - Marque : nul - Modèle : nul - Couleur : nul - nbrPlace : nul **Clé étrangère :** - note_id : nul	- **Id** : clé primaire, auto-incrémentation, non nul - lieuDep : non nul - lieuArr : non nul - PI : nul - dateDep : non nul - horrDep : non nul - Prix : non nul - infoRDV : nul - contrainte : nul - nbrPlace : non nul **Clé étrangère :** - utilisateur_id : non nul - groupe_id : nul	- **Id** : clé primaire, auto-incrémentation, non nul - note : nul **Clé étrangère :** Conducteur_id : non nul Utilisateur_id : non nul Offre_id : non nul	- **Id** : clé primaire, auto-incrémentation, non nul - status : non nul - message : nul **Clé étrangère :** Offre_id : non nul Utilisateur_id : non nul	- **Id** : clé primaire, auto-incrémentation, non nul - nomGroupe : non nul