- RENDU SAE DE BDD -
- R3.07 -

BLONDEAU Nicolas – BRUNET Baptiste – CHAMPEAU Yann – EIRAS Cléo - CHALMIN Lola

Notre équipe, InProgress, a été missionnée par le Product Owner afin de réaliser une application web et mobile pour jouer au jeu de Kayles version graphes, NodeMunch. Le principe est simple : ce jeu se joue au minimum à deux, et, sur un graphe donné, il faut supprimer des nœuds, ce qui engendre d'autres actions. En effet, lorsque l'on détruit un nœud, cela détruit ses voisins et les arêtes qui y mènent. Le jeu s'arrête lorsque le graphe est vide et qu'un joueur ne peut plus détruire de nœud, ce qui, dans ce cas, le fait perdre. Nous avons donc besoin de stocker tous les différents graphes dans la base de données ainsi que les informations concernant les utilisateurs. Pour ce faire, nous avons analysé ce problème et en avons déduit le Modèle Conceptuel de Données, représenté sur la Figure 1. Nous l'avons ensuite transformé en Modèle Logique de Données qui est représenté sur la Figure 2.

Dans notre MCD nous retrouvons donc les principales classes que représentent Graph et User. Nous avons décidé de représenter un Graph de cette manière car il sera composé de nœuds et de liens. On peut voir que le Graph est bien relié à Node par sa composition, en revanche Edge n'y est pas relié. Ce choix se justifie par le fait de ne pas répéter les attributs dans une entité liant Graph et Edge qui ne nous serait pas si utile. En effet, dans un souci d'optimisation et pour éviter la répétition du stockage des nœuds dans la base de données, nous avons décidé que nous rechercherions directement les liens existants entre les nœuds qui composent le graphe. Nos requêtes seront peut-être un peu plus longues si le graphe est composé de beaucoup de nœuds. En revanche nous pensons que cette méthode est meilleure et moins coûteuse que le stockage dans la base de données d'une entité Edge associée à un Graph et ce, pour chaque Edge de chaque Graph. De plus, nous avons décidé de donner à chaque classe un identifiant ce qui permettra d'avoir une clé primaire unique à toute classe. Ceci sera également plus léger qu'un varchar par exemple lorsqu'il s'agira de stocker cet identifiant en tant que clé étrangère.

Nous avons également choisi de stocker une image de notre graphe qui servira de prévisualisation au sein de l'application avant de choisir un graphe. Ceci permettra de voir quel est le type du graphe et potentiellement ce qu'il est possible de faire comme coup ou autre. C'est donc ce que représente notre entité Thumbnail sur notre MCD.

Nous avons également un User qui a ses caractéristiques, celles-ci auront une valeur par défaut pour les utilisateurs non connectés. Nous avons également une table que nous avons appelé GroupUser, qui définira quel est le rôle d'un utilisateur. Ce statut sera soit un utilisateur lambda, soit un utilisateur connecté ou bien un administrateur. Ainsi, il nous sera plus facile de modérer le système et l'application grâce aux rôles attribués à chaque User. Chaque User peut également avoir des amis, ce qui entraîne cette liaison réflexive.

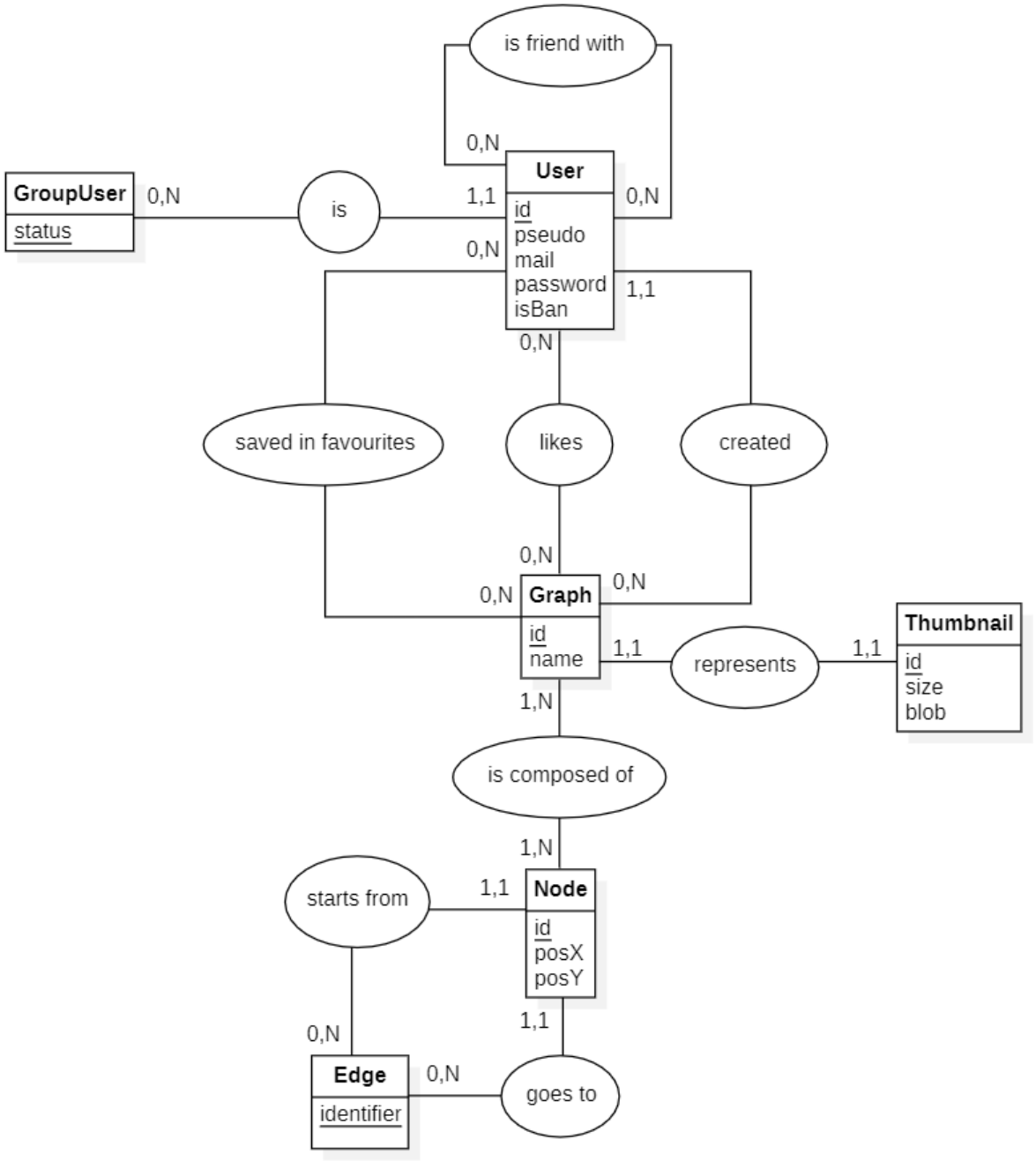
is friend with
0,N
User
id
pseudo
mail
password
isBan
0,N
1,1
0,N
GroupUser
status
0,N
is
1,1
0,N
1,1
0,N
saved in favourites
likes
created
0,N
0,N
Graph
id
name
0,N
1,1
represents
1,1
Thumbnail
id
size
blob
1,N
is composed of
1,N
starts from
1,1
Node
id
posX
posY
1,1
0,N
Edge
identifier
0,N
goes to

Figure 1 : MCD

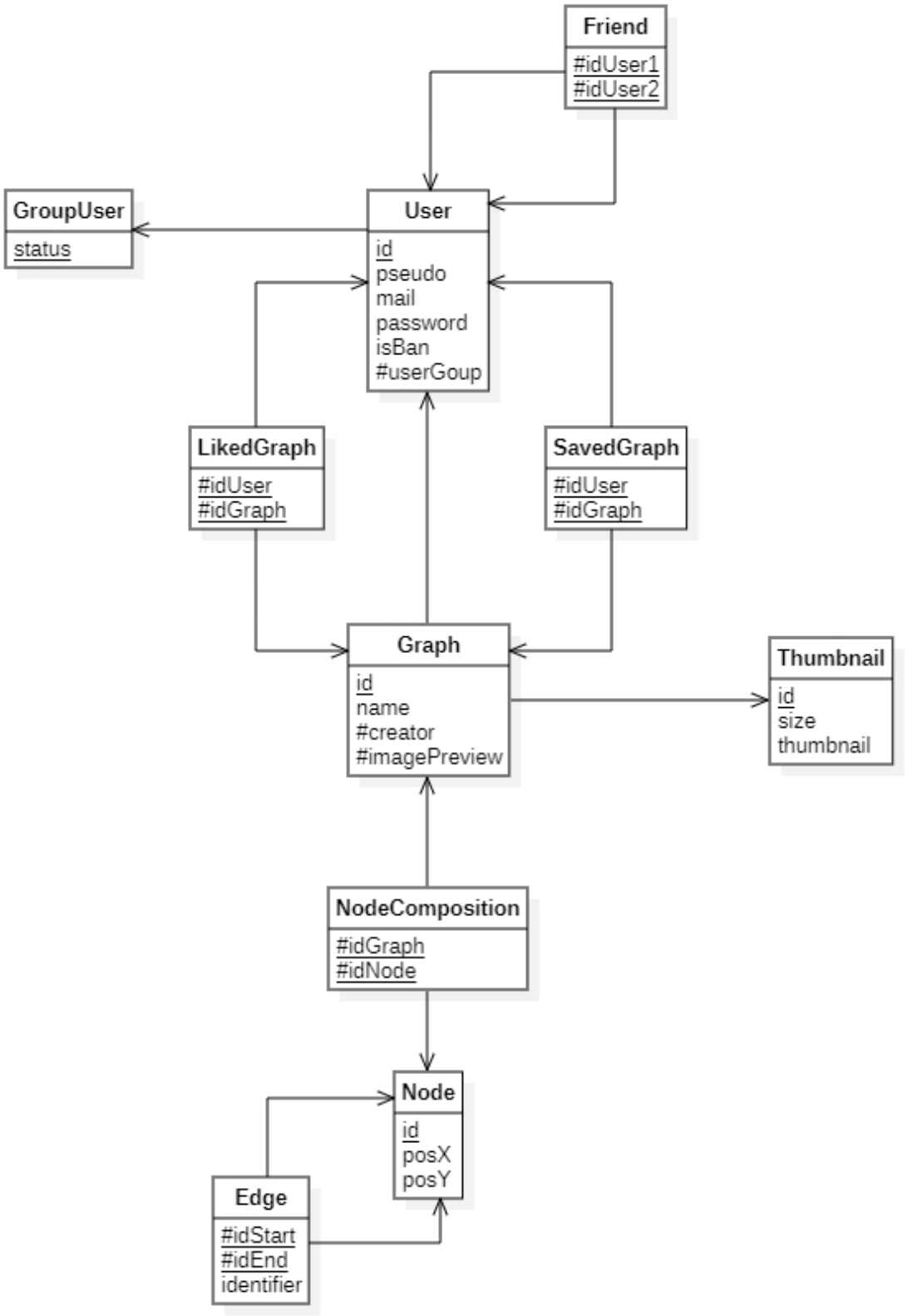

Figure 2 : le MLD