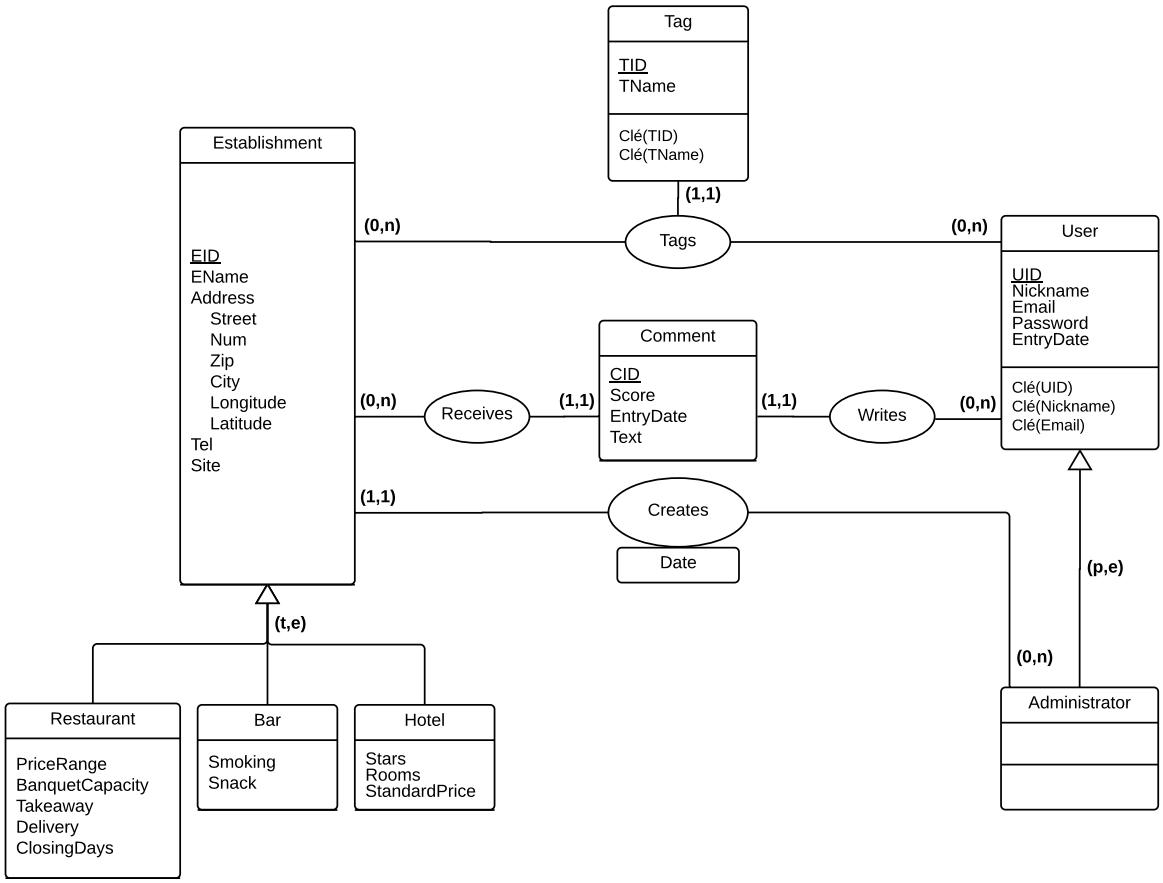
INFO-H-303 : Base de données
Projet : Annuaire d'établissements horeca

Frantzen Christian Küpper Marius

12 mai 2016

Modèle entité-association



https ://preview.overleaf.com/public/fmggjjhwnrrs/images/8586f5f52acf38bd1764f56ff49298182f11036b.jpeg

FIGURE 1 – Modèle entité-association

Contraintes d'intégrité

- Un *User* peut commenter plusieurs fois le même *Establishment* à des dates différentes.
- Un *User* ne peut pas apposer le même *Tag* plusieurs fois sur le même *Establishment*.
- L'*EntryDate* d'un *Adminisitrator* doit être strictement supérieure à l'*EntryDate* de l'*Establishment* qu'il a créé.
- L'*EntryDate* d'un *Comment* doit être strictement supérieure à l'*EntryDate* du User qui le fait ainsi que l'*EntryDate* de l'*Establishment* sur lequel il est fait.
- Un *User* ne peut pas donner deux likes/dislikes sur un *Comment*.

Remarques

Si un *Restaurant* ne veut pas organiser de banquet, il spécifie sa *BanquetCapacity* comme étant 0.

Modèle relationnel

Establishment(EID,EName,Street,Num,Zip,City,Longitude,Latitude,Tel,Site,UID,EntryDate,HorecaType)

— UID référence User.UID (représente le createur de l'établissement)

Restaurant(EID, PriceRange, BanquetCapacity, Takeaway, Delivery)
— EID référence Establishment.EID

RestaurantClosingDays(EID, ClosingDay, Hour)
— EID référence Establishment.EID

Bar(EID, Smoking, Snack)
— EID référence Establishment.EID

Hotel(EID, Stars, Rooms, StandardPrice)
— EID référence Establishment.EID

User(UID, Nickname, Email, Password, EntryDate, Admin, PictureName)
— Nickname est unique et donc également une clé de cette relation
— Email est unique et donc également une clé de cette relation

Comment(CID, UID, EID, EntryDate, Score, Text, Likes)
— UID référence User.UID
— EID référence Establishment.EID
— (UID,EID,EntryDate) est unique et donc également une clé de cette relation

CommentLikes(CID, UID, Likes)
— CID référence Comment.CID
— UID référence User.UID
— (CID,UID) est unique et donc une clé de cette relation

Tag(TID, TName)
— TName est unique et donc également une clé de cette relation

EstablishmentTag(TID, EID, UID)
— TID référence Tag.TID
— EID référence Establishment.EID
— UID référence User.UID
— (TID,EID,UID) est unique et donc également une clé de cette relation

Remarques

L'ajout d'une entrée dans *Establishment* implique l'ajout d'une entrée dans soit *Restaurant*, soit *Bar* ou soit *Hotel*. Les deux entrées ont le même *EID*.

Contraintes de domaine

— $User.Admin \in \{True, False\}$ (Seul les *User* avec *User.Admin* = True ont les droits de créer, supprimer ou modifier des *Establishment*).
— Pour les entrées dans *Comment*, *Comment.EntryDate* > *Establishment.EntryDate* et *Comment.EntryDate* > *User.EntryDate*
— $Comment.Score \in \{0, 1, 2, 3, 4, 5\}$
— $Hotel.Stars \in \{0, 1, 2, 3, 4, 5\}$

— $Hotel.Rooms \in \mathbb{N}_0$
— $Hotel.StandardPrice \in \mathbb{N}_0$
— $Restaurant.BanquetCapacity \in \mathbb{N}$
— $Restaurant.PriceRange \in \mathbb{N}_0$
— $Restaurant.Takeaway \in \{True, False\}$
— $Restaurant.Delivery \in \{True, False\}$
— $RestaurantClosingDays.Hour \in \{"AM","PM","COMPLETE"\}$ (matin / après-midi / jour entier)
— $RestaurantClosingDays.ClosingDay \in \{"MON","TUE","WED","THU","FRI","SAT","SUN"$
— $Bar.Smoking \in \{True, False\}$
— $Bar.Snack \in \{True, False\}$
— $Comment.Likes \in \mathbb{Z}$
— $CommentLikes.Likes \in \{True, False\}$

Requêtes

R1

Tous les utilisateurs qui apprécient au moins 3 établissements que l'utilisateur "Brenda" apprécie.

Calcul relationnel tuple

$$\{u | User(u) \wedge \exists e_1 \exists e_2 \exists e_3 (Establishment(e_1) \wedge Establishment(e_2) \wedge Establishment(e_3) \wedge e_1 \neq e_2 \wedge e_1 \neq e_3 \wedge e_2 \neq e_3 \wedge \exists u_2 \exists c_1 \exists c_2 (User(u_2) \wedge Comment(c_1) \wedge Comment(c_2) \wedge u_2.uid \neq u.uid \wedge u_2.nickname = "Brenda" \wedge \forall i \in \{1,2,3\} (c_1.uid = e_i \wedge c_2.uid = e_i \wedge c_1.uid = u.uid \wedge c_2.uid = u_2.uid \wedge c_1.score \geq 4 \wedge c_2.score \geq 4)))\}$$

Algèbre relationnelle

$$EstabsBrendaLikes \leftarrow \pi_{eid}((\sigma_{score \geq 4}(Comment)) *_{uid=uid} (\sigma_{nickname='Brenda'}(User)))$$
$$EstabsOthersLike \leftarrow \pi_{comment.eid,\ user.*}((\sigma_{score \geq 4}(Comment)) *_{uid=uid} (\sigma_{nickname \neq 'Brenda'}(User)))$$
$$Result \leftarrow \sigma_{nbrLikes \geq 3}(user.*, nbrLikes \quad \gamma \quad count(eid)\ as\ nbrLikes ((EstabsBrendaLikes) *_{eid=eid} (EstabsOthersLike)))$$

SQL

```
SELECT u1.*
FROM users u1, comments c1
WHERE u1.nickname != 'Brenda' AND c1.uid = u1.uid AND c1.score >= 4 AND c1.eid IN (
        SELECT DISTINCT c2.eid
        FROM comments c2, users u2
        WHERE c2.score >= 4 AND u2.nickname = 'Brenda' AND c2.uid = u2.uid
        )
GROUP BY u1.uid
HAVING COUNT( DISTINCT c1.eid ) >= 3
```

R2

Tous les établissements qu'apprécie au moins un utilisateur qui apprécie tous les établissements que "Brenda" apprécie.

Calcul relationnel tuple

$$\{e|Establishment(e) \wedge \exists u_1 \exists c_1(User(u_1) \wedge Comment(c_1) \wedge c_1.eid = e.eid \wedge c_1.uid = u_1.uid \wedge u_1.nickname \neq \text{"}Brenda\text{"} \wedge c_1.score \geq 4 \wedge \exists u_2 \exists c_2 \exists c_3 \exists e_2(User(u_2) \wedge Comment(c_2) \wedge Comment(c_3) \wedge Establishment(e_2) \wedge u_2.nickname = \text{"}Brenda\text{"} \wedge c_2.eid = u_2.uid \wedge c_2.eid = e_2.eid \wedge c_3.eid = e_2.eid \wedge c_3.uid = u_1.uid \wedge c_2.uid = u_2.uid \wedge c_2.score \geq 4 \wedge c_3.score \geq 4))\}$$

Algèbre relationnelle

$$HighscoreComments \leftarrow \sigma_{score \geq 4}(Comment)$$
$$EstabsBrendaLikes \leftarrow \pi_{eid}((\sigma_{nickname=Brenda}(User)) *_{uid=uid} (HighscoreComments))$$
$$EstabsOthersLike \leftarrow \pi_{eid,uid}((\sigma_{nickname \neq Brenda}(User)) *_{uid=uid} (HighscoreComments))$$
$$LikesSameAsBrenda \leftarrow \pi_{uid}(EstabsOthersLike / EstabsBrendaLikes)$$
$$Result \leftarrow \pi_{estab.*}((Establishment) *_{eid=eid} ((LikesSameAsBrenda) *_{uid=uid} (EstabsOthersLike)))$$

SQL

```
SELECT DISTINCT e1.*
FROM users u1, establishments e1, comments c1
WHERE u1.uid = c1.uid AND u1.nickname != 'Brenda' AND e1.eid = c1.eid AND c1.score >= 4 AND
      EXISTS (
          SELECT DISTINCT c2.*
          FROM comments c2
          WHERE u1.uid = c2.uid AND c2.score >= 4 AND c2.uid = u1.uid AND e1.eid = c2.eid AND
              SELECT c2.eid
              FROM comments c3, users u2
              WHERE c3.score >= 4 AND u2.nickname = 'Brenda' AND c3.uid = u2.uid AND c3.eid =
          )
      )
```

R3

Tous les établissements pour lesquels il y a au plus un commentaire.

Calcul relationnel tuple

$$\{e|Establishment(e) \wedge \exists! c_1(Comment(c_1) \wedge c_1.eid = e.eid) \vee \nexists c_2(Comment(c_2) \wedge c_2.eid = e.eid)\}$$

Algèbre relationnelle

$$0\_Comments \leftarrow (Establishment) - ((Establishment) *_{eid=eid} (\pi_{eid}(Comment)))$$
$$1\_Comment \leftarrow ((Establishment) *_{eid=eid} (\sigma_{nbrComment=1}(eid, nbrComment \quad \gamma \quad count(cid) \text{ as } nbrComment(\pi_{eid,cid}(Comment)))))$$
$$Result \leftarrow 0\_Comments \cup 1\_Comment$$

SQL

```
SELECT e1.*
FROM establishments e1
WHERE NOT EXISTS (
```

```
    SELECT c1.*
    FROM comments c1
    WHERE e1.eid = c1.eid
    ) OR e1.eid IN (
        SELECT c2.eid
        FROM comments c2
        WHERE c2.eid = e1.eid
        GROUP BY e1.eid
        HAVING COUNT( c2.cid ) = 1
    )
```

R4

La liste des administrateurs n'ayant pas commenté tous les établissements qu'ils ont crées.

Calcul relationnel tuple

$$\{u | User(u) \wedge u.admin = true \wedge \exists e(Establishment(e) \wedge e.uid = u.uid \wedge \nexists c(Comment(c) \wedge c.eid = e.eid \wedge c.uid = u.uid))\}$$

Algèbre relationnelle

$$Admins \leftarrow \sigma_{admin=1}(User)$$
$$AdminComments \leftarrow \pi_{uid,eid}((Admins) *_{uid=uid} (Comment))$$
$$AdminCreated \leftarrow \pi_{uid,eid}((Admins) *_{uid=uid} (Establishment))$$
$$Result \leftarrow \pi_{user.*}((User) *_{uid=uid} (AdminCreated - AdminComments))$$

SQL

```
SELECT u1.*
FROM users u1
WHERE u1.admin = 1 AND EXISTS (
    SELECT e1.*
    FROM establishments e1
    WHERE u1.uid = e1.uid AND NOT EXISTS (
        SELECT c1.*
        FROM comments c1
        WHERE c1.eid = e1.eid AND c1.uid = u1.uid
    )
)
```

R5

SQL

```
SELECT e1.*, AVG( c1.score ) AS _avg
FROM establishments e1, comments c1
WHERE e1.eid = c1.eid
GROUP BY e1.eid
HAVING COUNT( DISTINCT c1.cid ) >= 3
ORDER BY AVG( c1.score ) DESC
```

R6

SQL

```
SELECT t.*, AVG( estab_score._avg ) AS score_avg
FROM tags t
INNER JOIN (
    SELECT et2.tid AS _tid, et2.eid AS _eid, AVG(c.score) AS _avg
    FROM comments c, establishment_tags et2
    WHERE c.eid = et2.eid AND et2.tid IN (
        SELECT et3.tid AS _tid
        FROM establishment_tags et3
        GROUP BY et3.tid
        HAVING COUNT( DISTINCT et3.eid ) >= 5
    )
    GROUP BY et2.eid
) AS estab_score ON estab_score._tid = t.tid
GROUP BY t.tid
ORDER BY AVG( estab_score._avg ) DESC
```

Hypothèses

- Deux *User*s ne peuvent pas avoir le même *Nickname* : L'interface doit permettre de consulter la fiche de chaque *User*. Un *User* qui cherche le profil de quelqu'un ne peut pas distinguer deux *User*s avec le même *Nickname* puisque le celui-ci est la seule donnée visible (pas d'image de profil, etc ...).
- Un *Admin* ne peut pas modifier des *Comment*s ni des *Tag*s : On pourrait lui donner les droits de gestion pour vérifier qu'il n'y ait pas d'abus, mais pour le moment un *Admin* ne peut que créer, modifier et enlever des *Establishment*s.
- Pour les requêtes R1 et R2 on exclut l'utilisateur "Brenda" comme étant un résultat qu'on considère lors de la recherche des résultats parce que ça nous semble un peu redondant d'.

Scénario

Étapes :

1. Login avec le compte de "Brenda"
2. Inspection de la page de son profile
3. Recherche des établissements avec le mot 'Be'
4. Inspection du profile du Restaurant 'Mirabelle'
5. Redaction d'un commentaire pour le restaurant
6. Ajout d'un label au restaurant qui n'existait pas encore
7. Logout
8. Login avec le compte de "Fred"
9. Ajout d'une photo de profile pour Fred
10. Création d'un Hotel
11. Inspection du profile du nouveau Hotel
12. Ajout d'un label au nouveau Hotel

13. Recherche et inspection du café 'Tavernier'
14. Ouverture de son site web
15. Supprimer le café Tavernier
16. Recherche et inspection du restaurant 'Chez Théo'
17. Changement de sa capacité de banquet et de ses heures d'ouvertures
18. Logout
19. Création d'un nouveau compte
20. Login avec le nouveau compte
21. Inspection compte de 'Fred'
22. Affichage des résultats des requêtes R1-R6
23. Logout

Instructions d'installation de notre application

Version de PHP utilisé : PHP 5.6.16

1. placer le dossier "ba3BDD" dans le dossier cible de votre serveur. ('www' pour wamp)
2. ajouter la base de donnée en important le fichier 'Create_Annuaire.sql'
3. Créez un compte d'utilisateur sur votre serveur avec le nom "projetChristianMarius" et le mot de passe "123soleil"
4. Importer les données XML en visitant "localhost/ba3BDD/Models/XMLfile_parser". Si rien ne s'affiche, tout c'est bien passé.
5. visitez "localhost/ba3BDD" et vous pouvez utiliser le site.

Les utilisateurs du fichier XML ont comme mot de passe leur nom d'utilisateur.