UNIVERSITÉ LIBRE DE BRUXELLES,
UNIVERSITÉ D'EUROPE

ULB

Faculté des Sciences
Département d'Informatique

INFO-H-303 Projet Base de données

Omer Nicolas
Rosette Arnaud

Année académique 2013 - 2014

Contents

1 Étude de cas

Il nous est demandé d'implémenter une application de création de flux d'information RSS 2.0. Il devra être possible pour des utilisateurs de consulter ces flux d'informations (par le biais de publications) et de s'abonner aux flux qui les intéressent. Ce concept est directement inspiré du lecteur de flux *Google Reader*. De plus, les utilisateurs pourront s'envoyer des demandes d'amitié (afin de s'abonner au flux de leur amis) et partager les publications qui les intéressent. Ils pourront également commenter les publications partagées s'ils le désirent.

2 Scénario

Pour cette démonstration, nous possédons une base de données contenant déjà certains utilisateurs, flux et publications. Nous commençons par ajouter un utilisateur en base base de données en sélectionnant le bouton d'inscription. Nous montrons qu'il n'est pas possible d'inscrire un utilisateur avec une adresse mail non valide ou avec une adresse mail déjà existante en base de données (l'adresse mail étant la clef d'identification). Après l'inscription, nous nous connectons avec ce nouvel utilisateur en entrant son mot de passe associé.

Une fois dans l'application, nous montrons l'inscription à un flux en entrant l'adresse url de ce dernier. Il est également possible d'ajouter un flux qui existe déjà en base de données à sa liste de souscription, pour cela il faut entrer l'url de ce flux. Une fois un flux ajouté et ses publications s'affichant dans la fenêtre correspondante, nous pouvons consulter les 10 dernières publications de ce flux soit dans l'application soit dans le navigateur internet par défaut.

Lorsqu'une publication a été lue, la case correspondante *lue* est cochée afin de se souvenir quelles publications ont été consultées. L'utilisateur peut aussi, en faisant un clic-droit sur la publication, partager la publication dans son flux personnel et la commenter s'il le désire (mais une seule fois seulement). Enfin, il est possible de télécharger les nouvelles publications d'un flux en cliquant sur le bouton correspondant.

Nous envoyons une demande en ami à un utilisateur existant (il n'est cependant pas pas possible de se demander soi-même en ami). On ferme l'application et on se connecte avec le récepteur de la demande d'ami. Si nous essayons d'envoyer à notre tour une demande à l'émetteur, l'application nous signale que c'est impossible. On accepte la demande en effectuant un clic droit sur la demande en attente pour l'accepter. Une fois la fenêtre principale de l'application rafraichie, on voit apparaître le flux du nouvel ami, nous pouvons à présent commenter ses publications.

3 Modèle entité-relation

Afin de modéliser la façon dont les informations seront physiquement enregistrées en base de données (au moyen d'SQL), voici un diagramme entité-association fournissant une description graphique du modèle conceptuel de données.

3.1 Schéma **(voir Figure 1)**

3.2 Hypothèses du modèle

La relation *FriendShip*, une fois traduite en base de données, comprend deux attributs mail_sender et mail_receiver (clefs étrangères référençant des champs de la table *User*). Ces derniers désignent respectivement l'émetteur et le récepteur de la demande d'amitié. On évite ainsi la confusion entre les deux et le cas problématique où l'émetteur de la demande d'amitié aurait accepté le demande sans l'aval du récepteur.

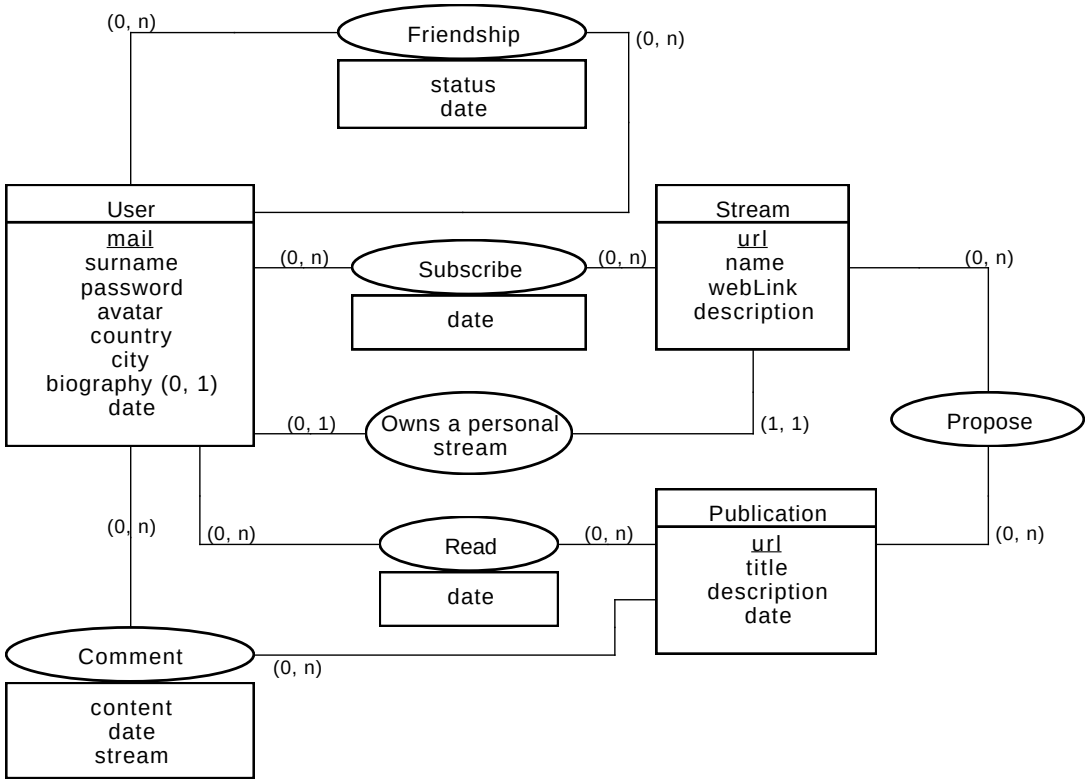

Figure 1: Modèle entité-relation de l'application

4 Modèle relationnel

Cette section comprend quant à elle le modèle logique directement tiré de la traduction du diagramme entité-association. Il s'agit également d'indiquer les références des sous-entités.

4.1 Traduction

User(mail, surname, password, avatar, country, city, *biography*, date, personal_stream_url)
personal_stream_url référence Stream.url

Stream(url, name, webLink, description)

Publication(url, title, date, description)

Friendship(mail_sender,mail_receiver, status, date)
mail_sender référence User.mail
mail_receiver référence User.mail

Comment(user_mail, publication_url, stream_url, content, date)
user_mail référence User.mail
publication_url référence Publication.url

Read(user_mail, publication_url, date)
user_mail référence User.mail

publication_url référence Publication.url

Propose(stream_url, publication_url)
stream_url référence Stream.url
publication_url référence Publication.url

Subscribe(user_mail, stream_url, date)
user_mail référence User.mail
stream_url référence Stream.url

4.2 Contraintes d'intégrité

- Un utilisateur ne peut commenter qu'une seule fois une publication.
- Un utilisateur peut commenter uniquement les publications de son flux personnel ou du flux d'un ami.
- Un utilisateur n'est pas inscrit à son propre flux.
- Un utilisateur ne peut pas envoyer une demande d'ami s'il a déjà reçu une demande du même utilisateur.
- Chaque utilisateur est inscrit au flux de ses amis.
- Sur une publication, un utilisateur ne voit que les commentaires de ses amis.
- La valeur de l'attribut *date* de la table *User* doit être antérieure aux valeurs respectives des attributs *date* de la relation *Subscribe*, *date* de la relation *Read*, *date* de la relation *Comment* et *date* de la relation *Friendship*.
- La valeur de l'attribut *date* de la table *Publication* doit être antérieure aux valeurs respectives des attributs *date* de la relation *Read* et date de la relation *Comment*.
- La valeur de l'attribut *mail* de la table *User* doit respecter un format valide, c'est à dire une expression du type : [a-zA-Z0-9+. -] + @[a-z0-9-.] + [a-z].
- La valeur des attributs *date* des tables *User*, *Publication*, *Friendship*, *Comment*, *Read* et *Subscribe* doivent respecter le format suivant : 'yyyy-mm-dd'.
- Un utilisateur ne peut pas s'ajouter lui-même en ami. Il est également impossible d'envoyer une demande à un utilisateur pour lequel il a déjà reçu une demande.

5 Justification des choix de modélisations

Nous avions pensé à créer une table *Personnal Stream* qui héritait de la table *Stream* mais cela compliquait inutilement le modèle. Dans le même ordre d'idées, il était possible d'avoir une table *Commented Publication* qui aurait hérité de *Publication* mais la relation *Comment* suffit amplement pour ce modèle. Notre application est disponible en français uniquement.

Nous avons utilisé le patron de conception Modèle Vue Contrôleur afin de correctement diviser les tâches de nos classes (les données, leur présentation et leur traitement). La vue (l'interface, les éléments avec lesquels l'utilisateur interagit) lit les données que lui fournit le modèle et manipule un contrôleur qui se charge de mettre à jour les données (gestion des événements).

Ajouté à cela, nous utilisons des objets d'accès au données qui sont instanciés à chaque interaction avec la base de données SQL. Ils se chargent d'insérer, supprimer, mettre à jour et chercher les données désirées. Nous préparons les requêtes à leur construction et nous remplissons les informations nécessaires pour l'interaction qu'ils ont avec la base de données.

Pour terminer, nous avons également des *parsers* de données xml afin de peupler notre base de données à partir d'un fichier avec l'extension correspondante.

6 Requêtes demandées

Cette section présente les six requêtes obligatoires écrites respectivement en forme SQL, en algèbre relationnelle et en calcul relationnel tuple. Cependant, seulement les requêtes 5 et 6 ne doivent pas figurer sous ces deux dernières formes.

1. Tous les utilisateurs qui ont au plus 2 amis
2. La liste des flux auxquels a souscrit au moins un utilisateur qui a souscrit à au moins deux flux auxquels X a souscrit
3. La liste des flux auxquels X a souscrit, auxquels aucun de ses amis n'a souscrit et duquel il n'a partagé aucune publication
4. La liste des utilisateurs qui ont partagé au moins 3 publications que X a partagé
5. La liste des flux auquel un utilisateur est inscrit avec le nombre de publications lues, le nombre de publications partagées, le pourcentage de ces dernières par rapport aux premières, cela pour les 30 derniers jours et ordonnée par le nombre de publications partagées.
6. La liste des amis d'un utilisateur avec pour chacun le nombre de publications lues par jour et le nombre d'amis, ordonnée par la moyenne des lectures par jour depuis la date d'inscription de cet ami

6.1 Requêtes SQL

1. ```
   SELECT u.*
   FROM User u LEFT JOIN (SELECT * FROM Friendship f1 WHERE f1.status = TRUE) AS f2
   ON f2.mail_sender = u.mail OR f2.mail_receiver = u.mail
   GROUP BY u.mail
   HAVING COUNT(*) < 3
   ```

2. ```
   <user>
   SELECT DISTINCT s3.*
   FROM Subscribe sub3, Stream s3
   WHERE sub3.stream_url = s3.url AND sub3.user_mail IN
       (SELECT u1.mail
       FROM User u1, Stream s2, Subscribe sub2
       WHERE sub2.user_mail != <user>.mail
       AND sub2.user_mail = u1.mail
       AND sub2.stream_url = s2.url
       AND sub2.stream_url IN
           (SELECT sub1.stream_url
               FROM Subscribe sub1
               WHERE sub1.user_mail = <user>.mail)
       GROUP BY sub2.user_mail
       HAVING COUNT(*) >= 2)
   ```

3. ```
   <user>
   SELECT *
   FROM Stream s
   WHERE s.url IN
       (SELECT sub.stream_url
       FROM Subscribe sub1
       WHERE sub1.user_mail = <user>.mail)
   AND s.url NOT IN
   ```
   ```

```
    (SELECT DISTINCT sub.stream_url
    FROM Subscribe sub2
    WHERE sub2.user_mail IN
        (SELECT u.mail FROM User u WHERE u.mail IN
            (SELECT f1.mail_sender FROM Friendship f1
            WHERE f1.mail_receiver = <user>.mail AND f1.status = TRUE)
        OR u.mail IN
            (SELECT f2.mail_receiver FROM Friendship f2
            WHERE f2.mail_sender = <user>.mail AND f2.status = TRUE)))
AND s.url NOT IN
    (SELECT DISTINCT prop2.stream_url
    FROM Propose prop2
    WHERE prop2.publication_url IN
        (SELECT prop1.publication_url
        FROM Publication pub, Propose prop1
        WHERE pub.url = prop1.publication_url
        AND prop1.stream_url = <user>.personal_stream_url))
```

4.
```
<user>
SELECT DISTINCT u.*
FROM User u, Propose prop1, Propose prop2
WHERE prop1.stream_url = <user>.personal_stream_url
AND prop2.stream_url != <user>.personal_stream_url
AND prop2.stream_url = u.personal_stream_url
AND prop1.publication_url = prop2.publication_url
GROUP BY u.mail
HAVING count(*) >= 3
```

5.
```
<user>
SELECT s1.url,
    @a:=(SELECT COUNT(*) FROM ‘Read‘ r2, Propose prop1
    WHERE r2.publication_url = prop1.publication_url
    AND r2.user_mail = <user>.mail AND prop1.stream_url = s1.url
    AND DATE_SUB(CURDATE(),INTERVAL 30 DAY) <= r2.date) AS publications_read,
    @b:=(SELECT COUNT(*) FROM Propose prop2, Propose prop3, Publication pub1
    WHERE  pub1.url = prop2.publication_url
    AND pub1.url = prop3.publication_url AND prop2.publication_url = prop3.publication_url
    AND prop2.stream_url = <user>.personal_stream_url AND prop3.stream_url = s1.url
    AND DATE_SUB(CURDATE(),INTERVAL 30 DAY) <= pub1.date) AS publications_shared,
    (@b / @a * 100)
FROM Stream s1, Subscribe sub1
WHERE sub1.stream_url = s1.url AND sub1.user_mail = <user>.mail
ORDER BY publications_shared
```

6.
```
<user>
SELECT u1.mail,
    ((SELECT COUNT(*) FROM ‘Read‘ r1 WHERE r1.user_mail = u1.mail) /
    (SELECT DATEDIFF(CURDATE(), u1.date))) AS publications_read_per_day,
    (SELECT COUNT(*) FROM User u2 WHERE u2.mail IN
    (SELECT friend3.mail_receiver FROM Friendship friend3
    WHERE friend3.mail_sender = u1.mail AND friend3.status = TRUE) OR u2.mail IN
    (SELECT friend4.mail_sender FROM Friendship friend4
    WHERE friend4.mail_receiver = u1.mail AND friend4.status = TRUE)) AS nb_friends
```

```
FROM User u1
WHERE
u1.mail IN (SELECT friend1.mail_receiver FROM Friendship friend1
WHERE friend1.mail_sender = <user>.mail AND friend1.status = TRUE)
OR u1.mail IN (SELECT friend2.mail_sender FROM Friendship friend2
WHERE friend2.mail_receiver = <user>.mail AND friend2.status = TRUE)
ORDER BY publications_read_per_day
```

6.2 Algèbre relationnelle

1.

$$\underset{2}{F} \leftarrow \sigma_{status=True}(Friendship) \tag{1}$$

$$T \leftarrow User \sqsupset\bowtie_{<mail\_sender=mail \vee mail\_receiver=mail>} \underset{2}{F} \tag{2}$$

$$Result \leftarrow \pi_{*}(mail \quad \sigma_{COUNT(*)<3}(T)) \tag{3}$$

2.

$$S1 \leftarrow \pi_{stream\_url}(\sigma_{user\_mail=<user>.mail}(Subscribe))` \tag{4}$$

$$a \leftarrow (\sigma_{user\_mail!=<user>.mail \wedge user\_mail=mail}(User * Subscribe)) \tag{5}$$

$$b \leftarrow (\sigma_{stream\_url=url \wedge stream\_url \ \cap S1}(Stream * Subscribe)) \tag{6}$$

$$U1 \leftarrow \pi_{mail}(user\_mail \quad \sigma_{COUNT(*)>=2}(a \wedge b)) \tag{7}$$

$$Result \leftarrow \pi_{*}(\sigma_{stream\_url=url \wedge user\_mail \cap U1}(Subscribe * Stream)) \tag{8}$$

3.

$$Sub1 \leftarrow \pi_{stream\_url}(\sigma_{user\_mail=<user>.mail}(Subscribe)) \tag{9}$$

$$F1 \leftarrow \pi_{mail\_sender}(\sigma_{mail\_receiver=<user>.mail \wedge status=true}(Friendship)) \tag{10}$$

$$F2 \leftarrow \pi_{mail\_sender}(\sigma_{mail\_sender=<user>.mail \wedge status=true}(Friendship)) \tag{11}$$

$$U \leftarrow \pi_{mail}(\sigma_{(mail \cap F1) \vee (mail \cap F2)}(User)) \tag{12}$$

$$Sub2 \leftarrow \pi_{stream\_url}(\sigma_{user\_mail \cap U}(Subscribe)) \tag{13}$$

$$Prop1 \leftarrow \pi_{publication\_url}(\sigma_{url=publication\_url \wedge stream\_url=<user>.personal\_stream\_url}(Propose * Publication)) \tag{14}$$

$$Prop2 \leftarrow \pi_{stream\_url}(\sigma_{publication\_url \cap Prop1}(Propose)) \tag{15}$$

$$Result \leftarrow \pi_{*}(\sigma_{(url \cap Sub1) \wedge (url \setminus Sub2) \wedge (url \setminus Prop2)}(Stream)) \tag{16}$$

4.

$$PP \leftarrow \pi_{*}(\sigma_{stream\_url=<user>.personal\_stream\_url}(Propose)) \tag{17}$$

$$PU \leftarrow \pi_{*}(\sigma_{(stream\_url=personal\_stream\_url) \wedge (stream\_url!=<user>.personal\_stream\_url)}(Propose * User)) \tag{18}$$

$$Result \leftarrow \pi_{*}(mail \quad \sigma_{COUNT(*)>=3}(PP \cap PU)) \tag{19}$$

6.3 Calcul relationnel tuple

1.

$$[u.*|\ User(u)\ \wedge \forall\ u_{2}(User(\ u_{2})\rightarrow((\forall f(Friendship(f)\rightarrow(f.status=False\wedge(f.mail\_sender=\ u_{2}.mail \quad (20)$$

$$\vee f.mail\_receiver=\ u_{2}.mail))))\vee(\exists\ f_{2}(Friendship(\ f_{2})\rightarrow(\ f_{2}.status=True\wedge(\ f_{2}.mail\_sender=\ u_{2}.mail \quad (21)$$

$$\vee\ f_{2}.mail\_receiver=\ u_{2}.mail))))\vee(\exists\ f_{3}\ f_{4}(F(\ f_{3})\rightarrow(\ f_{3}.status=True\wedge(\ f_{3}.mail\_sender=\ u_{2}.mail \quad (22)$$

$$\vee\ f_{3}.mail\_receiver=\ u_{2}.mail)))\wedge Friendship(\ f_{4})\rightarrow(\ f_{4}.status=True\wedge(\ f_{4}.mail\_sender=\ u_{2}.mail \quad (23)$$

$$\vee\ f_{4}.mail\_receiver=\ u_{2}.mail))\wedge(\ f_{3}\ !=\ f_{4})))))] \quad (24)$$

2.

$$[s.*|\ Stream(s)\ \wedge\forall u\forall sub_{1}\forall sub_{4}\exists\ s_{2}\exists\ s_{3}\exists sub_{2}\exists sub_{3}((User(u)\wedge Subscribe(sub_{1})\wedge Subscribe(sub_{2}) \quad (25)$$

$$\wedge Subscribe(sub_{3})\wedge Subscribe(sub_{4})\wedge Stream(\ s_{2})\wedge Stream(\ s_{3}))\rightarrow(((sub_{4}.stream\_url=s.url \quad (26)$$

$$\wedge\ sub_{4}.user\_mail)\cap((sub_{2}.user\_mail\ !=<user>.mail\wedge sub_{2}.user\_mail=u.mail \quad (27)$$

$$\wedge\ sub_{2}.stream\_url=\ s_{2}.url\wedge sub_{2}.stream\_url)\cap(sub_{1}.user\_mail=<user>.mail)) \quad (28)$$

$$\wedge(sub_{4}.stream\_url=s.url\wedge sub_{4}.user\_mail)\cap((sub_{3}.user\_mail\ !=<user>.mail \quad (29)$$

$$\wedge\ sub_{3}.user\_mail=u.mail\wedge sub_{3}.stream\_url=\ s_{3}.url\wedge sub_{3}.stream\_url) \quad (30)$$

$$\cap(sub_{1}.user\_mail=<user>.mail))\wedge(sub_{3}\ !=sub_{4})\wedge(\ s_{2}\ !=\ s_{3})))] \quad (31)$$

3.

$$[s.*|\ Stream(s)\ \wedge\forall u\ \forall f\ \forall\ s_{2}\ \forall\ s_{3}((Friendship(f)\wedge Subscribe(\ s_{2})\wedge Subscribe(\ s_{3})\wedge User(u))\rightarrow \quad (32)$$

$$(((u.mail=f.mail\_sender\wedge<user>.mail=f.mail\_receiver\wedge f.status=TRUE)\vee \quad (33)$$

$$(u.mail=f.mail\_receiver\wedge<user>.mail=f.mail\_sender\wedge f.status=TRUE))\wedge \quad (34)$$

$$(\ s_{2}.user\_mail=<user>.mail)\wedge(\ s_{3}.user\_mail=u.mail)\wedge(s\cap\ s_{2}=\varnothing)\wedge(s\cap\ s_{3}=\varnothing)))] \quad (35)$$

4.

$$[u.*|\ User(u) \wedge \forall\ u_2(User(\ u_2) \rightarrow (\exists\ p_1\ p_2\ p_3\ p_4\ p_5\ p_6\ (Proposition(\ p_1) \wedge Proposition(\ p_2) \wedge Proposition(\ p_3) \tag{36}$$

$$\wedge Proposition(\ p_4) \wedge Proposition(\ p_5) \wedge Proposition(\ p_6) \wedge\ p_1.stream\_url =< user > .personal\_stream\_url \tag{37}$$

$$\wedge\ p_2.stream\_url =< user > .personal\_stream\_url \wedge\ p_3.stream\_url =< user > .personal\_stream\_url \tag{38}$$

$$\wedge\ p_4.stream\_url = u.personal\_stream\_url \wedge\ p_5.stream\_url = u.personal\_stream\_url \tag{39}$$

$$\wedge\ p_6.stream\_url = u.personal\_stream\_url \wedge\ p_1.publication\_url =\ p_4.publication\_url \tag{40}$$

$$\wedge\ p_2.publication\_url =\ p_5.publication\_url \wedge\ p_3.publication\_url =\ p_6.publication\_url \tag{41}$$

$$\wedge\ p_4.stream\_url\ ! =< user > .personal\_stream\_url \wedge\ p_5.stream\_url\ ! =< user > .personal\_stream\_url \tag{42}$$

$$\wedge\ p_6.stream\_url\ ! =< user > .personal\_stream\_url)))] \tag{43}$$

7 Instructions d'exécution du programme

Concernant les détails techniques, notre programme est codé en Java, utilise des fonctionnalité de la version 1.7, l'interface utilise la librairie Swing de Java et nous employons MySQL comme base de données. Nous avons dû adjoindre un connecteur pour Mysql afin d'obtenir une connexion au serveur MySQL. Il nécessite donc de lancer le serveur mysql avant l'utilisation de l'application (nous avons choisi un nom d'utilisateur et un mot de passe par défaut dans l'application). Nous avons généré un exécutable jar de notre application qui se lance selon l'action désirée :

- `java -jar rss-manager`
  pour démarrer en mode normal
- `java -jar rss-manager add populate-db.xml`
  pour peupler la base de données
- `java -jar rss-manager delete`
  pour supprimer les tables existantes