UNIVERSITÉ LIBRE DE BRUXELLES
FACULTÉ DES SCIENCES
DÉPARTEMENT D'INFORMATIQUE

INFO-H-303 Bases de données : Rapport de projet – Partie 2

POSTULA Loïs
PICARD Simon

Année académique 2013 - 2014

Contents

1 Modélisation entités-relations

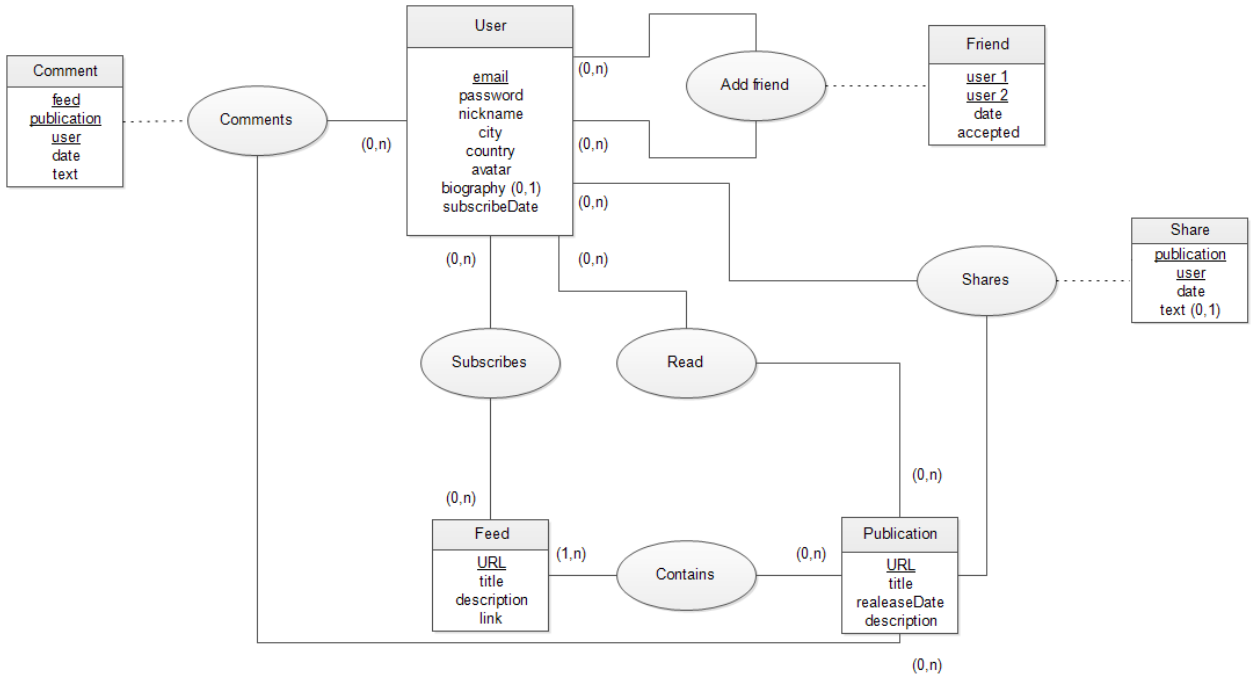

Figure 1: Modélisation entités-relations du projet.

2 Contraintes d'intégrité

1. Lorsqu'un utilisateur A désire effectuer une demande d'amitier il demande a un autre utlisateur B qui peut ou non accepter. Si utilisateur B accepte les utilisateurs sont amis. La demande ne peut être acceptée que par B si elle a été demandée par A.
2. La date d'inscription d'un utilisateur précède les dates de lectures, les dates de commentaires et les dates d'acceptation d'amis.
3. La date d'une publication précède les dates de lectrue et de commentaires de cette publication.
4. Les dates de lecture d'un publication précède les dates de commentaires de cette publication.
5. Les publication d'un flux utilisateur sont exactement les publications que cet utilisateur a partagées.
6. Un utilisateur ne peut partager ou lire que des publicatipns composant un flux qu'il suit.
7. Un utlisateur ne peut faire de commentaires que sur une publication partagée par un de ses amis (au moment du commentaire)
8. Un utilisateur ne peut pas être amis avec lui-même.
9. L'association d'amitié est bidirectionnelle. Si l'utilisateur A est ami de l'utilisateur B, alors B est ami de l'utilisateur A.
10. Un utilisateur suit les flux de ses amis.

3 Modélisation relationnelle

User (**email**, password, nickname, city, country, avatar, *biography*, subscription date.)

- seul le champ *biography* peut être nul
- *subscription date* ne peut être dans le futur

- *email* doit respecter le format d'e-mail
- *city* implique *pays*

Feed(**URL**, title, description, *link*)

- *URL* doit être une URL valide

Publication(**URL**, title, releaseDate, description)

- *releaseDate* ne peut être dans le futur

Comment(**user, publication, feed**, date, text)

- *user* référence *User.email*
- *publication* référence *Publication.URL*
- *feed* référence *Feed.URL*
- *date* ne peut être dans le futur

Friendship(**user_1, user_2**, accepted, date)

- *user_1* référence *User.email*
- *user_2* référence *User.email*
- *date* ne peut être dans le futur

Subscribes(**user, feed**, date)

- *user* référence *User.email*
- *feed* référence *Feed.URL*
- *date* ne peut être dans le futur

Read(**user, publication, feed**, date)

- *user* référence *User.email*
- *publication* référence *Publication.URL*
- *date* ne peut être dans le futur

Contains(**feed, publication**)

- *feed* référence *Feed.URL*
- *publication* référence *Publication.URL*

Share(**user, publication**, date, text)

- seul le champ *text* peut être nul
- *user* référence *User.email*
- *publication* référence *Publication.URL*
- *feed* référence *Feed.URL*
- *date* ne peut être dans le futur

4 Hypothèse

- Dans Friendship, user1 est celui qui a fait la demande d'ami et user2 celui qui la recoit.
- Dans Friendship, une requete avec le flag accepted à 0 est en attente de validation, à 1 elle est validée et si elle est refusé, elle supprimé de la table.
- Dans Friendship, si accepted vaut 0 alors le champ date vaut celle de la demande, s'il est à 1 alors le champs date représente la date d'ajout en amis.
- Dans la table Contain, il peut y avoir plusieur référence vers une même publication car cette derniere peut se trouver dans des fluxs personnels en plus de celui du rss de base.
- Un commentaire ne doit etre affiché que sur certains fluxs. En effet, l'utilisateur ne peut écrire un commentaire uniquement lorsqu'il l'a partagée ou si elle lui a été partagée. Comme il ne peut y avoir qu'un commentaire par publication d'un flux, ces trois éléments sont suffisants pour identifier un commentaire.

5 Choix d'implémentation

5.1 Language – Java

Nous avions comme contrainte que nous devions développer sur deux platforme différente, nous avons donc optez pour *Java* qui permet cela.

5.2 Librairie graphique – JavaFX 8

JavaFX 8 est la plus récente librairie graphique d'Oracle, elle incorpore toutes les avancés de Java 1.8 et est simple d'utilisation.

6 Instructions d'installation

Le projet se trouve dans une archive tar.gz, pour pouvoir l'exécuter, il faut avoir une base de donnée créé appelé "rssreader", avec un utilisateur "rssreader" et un mot de passe "rssreader". Le programme ne s'exécute que en Java 1.8, et il suffit alors de décompresser l'archive et d'excécuter:

java -jar rss_reader.jar

Comme il était au départ demandé de pouvoir parser un fichier XML pour peuplé la base de donnée, nous avons créer un générateur de de fichier XML, cependant ce dernier ne crée pas de commentaire de "read" sur des "share", pour la base de donnée peuplée, nous avons utiliser ce générateur et nous avons ajouté manuellement quelque commentaire sur le flux personnel de l'utilisateur Gothdor@gmail.com, dès lors nous vous conseillons de vous connecter avec celui ci (le mot de passe est "azerty").

7 Scénario de démonstration

- Creation d'un compte A
- Connexion de A
- Ajout d'un flux
- Ouvrir une publication
- Passer en mode lu
- Envoyer une demande d'amis à B
- Partager une publication
- Connexion B
- Accepter A
- Commenter publication de A

- Retour compte A, ajouter un deuxième flux
- Passer en mode toute les publications
- Modifier son avatar
- Rechercher un ami et montrer les requêtes
- Rechercher un flux et montrer les requêtes

8 Requêtes

8.1 Requête 1

Tous les utilisateurs qui ont au plus 2 amis.

8.1.1 SQL

```sql
SELECT * FROM user u
LEFT OUTER JOIN friendship f
ON
(
  f.user1_email = u.email
OR
  f.user2_email = u.email
)
GROUP BY u.email
HAVING COUNT(u.email) < 3
```

8.1.2 Algèbre relationnel

$$\begin{array}{lll} Request & \leftarrow & user \sqsupset\bowtie_{email=user_1\_email \vee email=user_2\_email} (friendship) \\ Friends & \leftarrow & \sigma_{accepted=TRUE}\ (Request) \end{array}$$

Il faut maintenant exclure tout ceux qui ont au moins 3 amis différents

$$\begin{array}{lll} f_1 & \leftarrow & \pi_{user_1\_email, f_1=user_2\_email}(Friends) \\ f_2 & \leftarrow & \pi_{user_1\_email, f_2=user_2\_email}(Friends) \\ f_3 & \leftarrow & \pi_{user_1\_email, f_3=user_2\_email}(Friends) \\ two\_friends & \leftarrow & f_1 *_{user_1\_email=user_1\_email} f_2 \\ three\_friends & \leftarrow & two\_friends *_{user_1\_email=user_1\_email} f_3 \\ three\_friends\_different & \leftarrow & \sigma_{f_1 \neq f_2 \neq f_3}(three\_friends) \\ under\_2\_friends & \leftarrow & \pi_{email}(user) - \pi_{email}(three\_friends\_different) \\ Result & \leftarrow & user *_{email\ =\ email}\ under\_2\_friends \end{array}$$

8.1.3 Calcul relationnel tuple

Définissons le prédicat *Friends(user1, user2)* comme étant:

$$\begin{aligned} Friends(u_1, u_2) : \exists\ d\ (Friendship(d) \wedge \\ (d.user_1\_email\ =\ u_1.email\ \vee\ d.user_2\_email\ =\ u_2.email) \wedge \\ (d.user_2\_email\ =\ u_1.email\ \vee\ d.user_1\_email\ =\ u_2.email) \wedge \\ d.accepted\ =\ 1 \end{aligned}$$

La requête devient alors:

$$\begin{aligned} u | User(u) \wedge \\ \exists friend_1((User(friend_1) \wedge Friends(u, friend_1)) \rightarrow \\ \exists friend_2((User(friend_2) \wedge friend_1 \neq friend_2 \wedge Friends(u, friend_2)) \rightarrow \\ \nexists friend_3((User(friend_3) \wedge friend_1 \neq friend_2 \neq friend_3 \wedge Friends(u, friend_3))))) \end{aligned}$$

8.2 Requête 2

La liste des flux auxquels a souscrit au moins un utilisateur qui a souscrit à au moins deux flux auxquel X a souscrit.

8.2.1 SQL

```
1  SELECT * FROM feed f
2  INNER JOIN feedsubscription c ON c.feed_url = f.url
3  INNER JOIN
4    (
5    SELECT b.user_email FROM feedsubscription b
6    INNER JOIN
7      (SELECT feed_url FROM feedsubscription WHERE user_email = <user>.email) a
8    ON a.feed_url = b.feed_url
9    GROUP BY b.user_email
10   HAVING COUNT(b.user_email) > 1
11   ) d
12 ON d.user_email = c.user_email
13 GROUP BY c.feed_url
```

8.2.2 Algèbre relationnel

$$
\begin{array}{lcl}
feed\_of\_ < \text{user} > & \leftarrow & \pi_{feed\_url}\ (\sigma_{user\_email=<user>.email}\ (feedsubscription)) \\
f\_as\_ < \text{user} >_1 & \leftarrow & \pi_{user\_email, feed_1=feed\_url}\ (feedsubscription) \\
f\_as\_ < \text{user} >_2 & \leftarrow & \pi_{user\_email, feed_2=feed\_url}\ (feedsubscription) \\
same\_feed\_sub & \leftarrow & f\_as\_ < \text{user} >_1 * (f\_as\_ < \text{user} >_2) \\
two\_different\_feed & \leftarrow & \pi_{user\_email}(\sigma_{feed_1 \neq feed_2}(same\_feed\_sub)) \\
feeds & \leftarrow & \pi_{feed\_url}(user *_{user.email=feedsubscription.user\_email} feedsubscription) \\
Result & \leftarrow & feeds * feed
\end{array}
$$

8.2.3 Calcul relationnel tuple

$$
\begin{aligned}
& f \mid \text{feed}(f) \wedge \\
& \exists\, fs_1\ (\text{feedsubscription}(fs_1) \wedge fs_1.feed\_url = f.url \wedge \\
& \exists\, fs_2\ (\text{feedsubscription}(fs_2) \wedge fs_2.user\_email = fs_1.user\_email \wedge \\
& \exists\, fs_3\ (\text{feedsubscription}(fs_3) \wedge fs_3.user\_email = fs_1.user\_email \wedge fs_3.feed\_url \neq fs_2.feed\_url \wedge \\
& \exists\, fs_4\ (\text{feedsubscription}(fs_4) \wedge fs_4.feed\_url = fs_2.feed\_url \wedge fs_4.user\_email = < \text{user} > .email \wedge \\
& \exists\, fs_5\ (\text{feedsubscription}(fs_5) \wedge fs_5.feed\_url = fs_3.feed\_url \wedge fs_3.user\_email = < \text{user} > .email))))
\end{aligned}
$$

8.3 Requête 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.

8.3.1 SQL

```
SELECT * FROM feed f
INNER JOIN feedsubscription fs1
  ON
  fs1.feed_url=f.url AND fs1.user_email = <user>.email
WHERE NOT EXISTS
  (
  SELECT c1.feed_url FROM contain c1
    INNER JOIN contain c2
      ON
      c2.publication_url = c1.publication_url
      AND
      c2.feed_url = "feed://" + <user>.email
  WHERE c1.feed_url=f.url
  )
AND NOT EXISTS
  (
  SELECT fs2.feed_url FROM feedsubscription fs2
    INNER JOIN friendship fr
    ON
      (
        fr.user1_email = fs2.user_email
        AND
        fr.user2_email = <user>.email
      )
      OR
      (
        fr.user2_email = fs2.user_email
        AND
        fr.user1_email = <user>.email
      )
      AND
        accepted = TRUE
   )
WHERE fs2.feed_url = f.url"
```

8.3.2 Algèbre relationnel

$$
\begin{array}{lcl}
Shared\_Feeds & \leftarrow & contain \bowtie_{publication\_url\_1=publication\_url\_2 \wedge feed\_url\_2=user.email} \ (contain) \\
Shared\_Feed\_X & \leftarrow & \pi_{feed\_url} \sigma_{feed\_url\_2=X} \ (Shared\_Feeds) \\
Friend\_Feeds & \leftarrow & feedsubscription \bowtie_{((user1\_email=email \wedge user2\_email=user.email)} \\
 & & \vee (user2\_email=email \wedge user2\_email=<user>.email)) \\
 & & \wedge accepted=TRUE \ (friendship) \\
Friend\_Feed\_X & \leftarrow & \pi_{feed\_url} \sigma_{feed\_url=X} \ (Friend\_Feeds) \\
User\_Feed & \leftarrow & feed \bowtie_{feed\_url=url \wedge user\_email=user.email} \ (feedsubscription) \\
Result & \leftarrow & (User\_Feed - Shared\_Feed\_X) - Friend\_Subscribed\_X
\end{array}
$$

8.3.3 Calcul relationnel tuple

$$
\begin{aligned}
& f|\ feed(f) \wedge \\
& (\ \exists\ s\ (feedsubscription(s) \wedge \\
& (\ s.feed\_url\ =\ f.url\ \wedge\ s.user\_email\ =\ <\text{user}>.email\ )) \wedge \\
& (\nexists\ c\ (contain(c)\ \wedge\ \exists\ d\ (contain(d)\ \wedge\ d.publication\_url = c.publication\_url \wedge \\
& d.feed\_url = \text{"feed://} + \text{"} <\text{user}>.email)) \wedge \\
& (\nexists\ e\ (feedsubscription(e)\ \wedge\ \exists\ fs\ (friendship(fs) \wedge \\
& ((fs.user1\_email\ =\ e.user\_email\ \wedge\ fs.user2\_email\ =\ <\text{user}>.email) \vee \\
& (fs.user2\_email\ =\ e.user\_email\ \wedge\ fs.user1\_email\ =\ <\text{user}>.email)) \wedge \\
& (fs.accepted\ =\ 1)))) \wedge \\
& (e.feed\_url = f.url)
\end{aligned}
$$

8.4 Requête 4

La liste des utilisateurs qui ont partagé au moins 3 publications que X a partagé.

8.4.1 SQL

```sql
SELECT * FROM user u
INNER JOIN sharedpublication s ON s.user_email = u.email
INNER JOIN
  (
  SELECT publication_url FROM sharedpublication
  WHERE user_email = <user>.email
  ) a
ON s.publication_url = a.publication_url
GROUP BY s.user_email
HAVING COUNT(s.user_email) > 2
```

8.4.2 Algèbre relationnelle

$$
\begin{array}{lcl}
Shared\_by\_ < user > & \leftarrow & \pi_{publication\_url} \left(\sigma_{user\_email=<user>.email} \left(sharedpublication \right) \right) \\
Reshared_1 & \leftarrow & \pi_{publication\_url=publication\_url_1, user\_email} \left(Shared\_by\_ < user > * sharedpublication \right) \\
Reshared_2 & \leftarrow & \pi_{publication\_url=publication\_url_2, user\_email} \left(Shared\_by\_ < user > * sharedpublication \right) \\
Reshared_3 & \leftarrow & \pi_{publication\_url=publication\_url_3, user\_email} \left(Shared\_by\_ < user > * sharedpublication \right) \\
Reshared & \leftarrow & (Reshared_1 * Reshared_2 * Reshared_3) \\
Reshared\_3\_times & \leftarrow & \pi_{user\_email} \left(\sigma_{publication\_url_1 \neq publication\_url_2 \neq publication\_url_3} \left(Reshared \right) \right) \\
Result & \leftarrow & user \bowtie_{email=user\_email} (Reshared\_3\_times)
\end{array}
$$

8.4.3 Calcul relationnel tuple

Définissons le prédicat *ResharedPublication(user1, user2)* comme étant:

$$
\begin{aligned}
& ResharedPublication(u_1, u_2) : \\
\exists\, s\, (Sharedpublication(publication\_url) \ \wedge s.user\_email \ = \ u_1.email) \wedge \\
\exists\, a\, (Sharedpublication(publication\_url) \ \wedge s.user\_email \ = \ u_2.email) \wedge \\
& a.publication\_url = s.publication\_url
\end{aligned}
$$

La requête devient alors:

$$
\begin{aligned}
& u \mid User(u) \wedge \\
\exists\, publication_1\, ((Sharedpublication(publication_1) \ \wedge \ & ResharedPublication(u, < user >)) \wedge \\
& \exists\, publication_2\, (Sharedpublication(publication_2) \wedge \\
& publication_1 \ \neq \ publication_2 \ \wedge \ ResharedPublication(u, < user >)) \wedge \\
& \exists\, publication_3\, (Sharedpublication(publication_3) \wedge \\
& publication_1 \ \neq \ publication_2 \ \neq \ publication_3 \\
& \wedge \ ResharedPublication(u, < user >)))
\end{aligned}
$$

8.5 Requête 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.

8.5.1 SQL

```
SELECT f.url, f.title, f.description, f.link, f.image,
(
  SELECT COUNT(*) FROM readstatus rs
  WHERE
    rs.feed_url = f.url
  AND
    rs.user_email = fs2.user_email
  AND
    TO_DAYS(NOW())-TO_DAYS(rs.date) < 30
)
AS nread,
(
  SELECT COUNT(*) FROM sharedpublication sp
  INNER JOIN feedsubscription fs
    ON
      sp.user_email = fs.user_email
  INNER JOIN contain c
    ON
        fs.feed_url = c.feed_url
      AND
        sp.publication_url = c.publication_url
  WHERE
    c.feed_url = f.url
  AND
    sp.user_email = fs2.user_email
  AND
    TO_DAYS(NOW())-TO_DAYS(sp.sharedDate) < 30
)
AS nshared,
(
  SELECT nshared/nread
)
AS ratio
FROM feed f
INNER JOIN feedsubscription fs2
ON
  fs2.feed_url = f.url
WHERE
  fs2.user_email = <user>.email
GROUP BY
  f.url
ORDER BY nshared
```

8.6 Requête 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

8.6.1 SQL

```
SELECT u.*,
(
  SELECT COUNT(*) FROM readstatus rs
  WHERE
  rs.user_email = u.email)/(TO_DAYS(NOW())-TO_DAYS(u.joinedDate)
)
AS mread,
(
  SELECT COUNT(u2.email) FROM user u2
  INNER JOIN friendship f2
  ON
      f2.user1_email = u2.email
    OR
      f2.user2_email = u2.email
  WHERE
  u2.email = u.email
  GROUP BY u2.email
)
AS nfriend
FROM user u
INNER JOIN friendship f
ON
    f.user1_email = u.email
  OR
    f.user2_email = u.email
WHERE
  (
    f.user1_email = <user>.email
  AND
    u.email <> f.user1_email
  )
OR
  (
    f.user2_email = <user>.email
  AND
    u.email <> f.user2_email
  )
ORDER BY mread
```

9 Script SQL DDL

```sql

SET SQL_MODE = "NO_AUTO_VALUE_ON_ZERO";
SET time_zone = "+00:00";


/*!40101 SET @OLD_CHARACTER_SET_CLIENT=@@CHARACTER_SET_CLIENT */;
/*!40101 SET @OLD_CHARACTER_SET_RESULTS=@@CHARACTER_SET_RESULTS */;
/*!40101 SET @OLD_COLLATION_CONNECTION=@@COLLATION_CONNECTION */;
/*!40101 SET NAMES utf8 */;

--
-- Bdd :  `rssreader`
--
CREATE DATABASE IF NOT EXISTS `rssreader` DEFAULT CHARACTER SET utf8 COLLATE
    utf8_bin;
USE `rssreader`;

-- --------------------------------------------------------

--
-- Structure de la table `comment`
--

CREATE TABLE IF NOT EXISTS `comment` (
  `feed_url` varchar(200) COLLATE utf8_bin NOT NULL,
  `publication_url` varchar(200) COLLATE utf8_bin NOT NULL,
  `user_email` varchar(200) COLLATE utf8_bin NOT NULL,
  `date` date NOT NULL,
  `text` text COLLATE utf8_bin NOT NULL,
  PRIMARY KEY (`feed_url`,`publication_url`,`user_email`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_bin;

-- --------------------------------------------------------

--
-- Structure de la table `contain`
--

CREATE TABLE IF NOT EXISTS `contain` (
  `feed_url` varchar(200) COLLATE utf8_bin NOT NULL,
  `publication_url` varchar(200) COLLATE utf8_bin NOT NULL,
  PRIMARY KEY (`feed_url`,`publication_url`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_bin;

-- --------------------------------------------------------

--
-- Structure de la table `feed`
--

CREATE TABLE IF NOT EXISTS `feed` (
  `url` varchar(200) COLLATE utf8_bin NOT NULL,
  `title` varchar(200) COLLATE utf8_bin NOT NULL,
  `link` varchar(200) COLLATE utf8_bin NOT NULL,
  `description` text COLLATE utf8_bin NOT NULL,
  `image` varchar(200) COLLATE utf8_bin DEFAULT NULL,
  PRIMARY KEY (`url`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_bin;
```

```
58
59 -- --------------------------------------------------------
60
61 --
62 -- Structure de la table `feedsubscription`
63 --
64
65 CREATE TABLE IF NOT EXISTS `feedsubscription` (
66   `user_email` varchar(200) COLLATE utf8_bin NOT NULL,
67   `feed_url` varchar(200) COLLATE utf8_bin NOT NULL,
68   `subscribedDate` date NOT NULL,
69   PRIMARY KEY (`user_email`,`feed_url`)
70 ) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_bin;
71
72 -- --------------------------------------------------------
73
74 --
75 -- Structure de la table `friendship`
76 --
77
78 CREATE TABLE IF NOT EXISTS `friendship` (
79   `user1_email` varchar(200) COLLATE utf8_bin NOT NULL,
80   `user2_email` varchar(200) COLLATE utf8_bin NOT NULL,
81   `requestDate` date NOT NULL,
82   `accepted` tinyint(1) NOT NULL,
83   PRIMARY KEY (`user1_email`,`user2_email`)
84 ) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_bin;
85
86 -- --------------------------------------------------------
87
88 --
89 -- Structure de la table `publication`
90 --
91
92 CREATE TABLE IF NOT EXISTS `publication` (
93   `url` varchar(200) COLLATE utf8_bin NOT NULL,
94   `title` varchar(200) COLLATE utf8_bin NOT NULL,
95   `releaseDate` date NOT NULL,
96   `description` text COLLATE utf8_bin NOT NULL,
97   `image` varchar(200) COLLATE utf8_bin DEFAULT NULL,
98   PRIMARY KEY (`url`)
99 ) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_bin;
100
101 -- --------------------------------------------------------
102
103 --
104 -- Structure de la table `readstatus`
105 --
106
107 CREATE TABLE IF NOT EXISTS `readstatus` (
108   `user_email` varchar(200) COLLATE utf8_bin NOT NULL,
109   `publication_url` varchar(200) COLLATE utf8_bin NOT NULL,
110   `feed_url` varchar(200) COLLATE utf8_bin NOT NULL,
111   `date` date NOT NULL,
112   PRIMARY KEY (`user_email`,`publication_url`,`feed_url`)
113 ) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_bin;
114
115 -- --------------------------------------------------------
116
117 --
118 -- Structure de la table `sharedpublication`
```

```
--

CREATE TABLE IF NOT EXISTS `sharedpublication` (
  `user_email` varchar(200) COLLATE utf8_bin NOT NULL,
  `publication_url` varchar(200) COLLATE utf8_bin NOT NULL,
  `sharedDate` date NOT NULL,
  `text` text COLLATE utf8_bin,
  PRIMARY KEY (`user_email`,`publication_url`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_bin;

-- --------------------------------------------------------

--
-- Structure de la table `user`
--

CREATE TABLE IF NOT EXISTS `user` (
  `email` varchar(200) COLLATE utf8_bin NOT NULL,
  `nickname` varchar(100) COLLATE utf8_bin NOT NULL,
  `password` varchar(100) COLLATE utf8_bin NOT NULL,
  `country` varchar(100) COLLATE utf8_bin NOT NULL,
  `city` varchar(100) COLLATE utf8_bin NOT NULL,
  `avatar` varchar(200) COLLATE utf8_bin NOT NULL,
  `biography` text COLLATE utf8_bin,
  `joinedDate` date NOT NULL,
  PRIMARY KEY (`email`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_bin;

/*!40101 SET CHARACTER_SET_CLIENT=@OLD_CHARACTER_SET_CLIENT */;
/*!40101 SET CHARACTER_SET_RESULTS=@OLD_CHARACTER_SET_RESULTS */;
/*!40101 SET COLLATION_CONNECTION=@OLD_COLLATION_CONNECTION */;
```