Programmation Web : FindLab

Ariane Odjo & Nourdine Bah

16 novembre 2014

Table des matières

1 Description et fonctionnalités **1**

2 Création de la base de donnée **2**

3 Requêtes **5**
3.1 Insertion 5
3.2 Interrogation 6

4 Informations complémentaires **7**

1 Description et fonctionnalités

Ce projet part du constat qu'il est souvent difficile d'avoir une vision globale des instituts, ou départements, travaillant sur une thématique de recherche précise. Nous avons créé une application qui géolocalise les instituts travaillant sur un sujet donné et les affiche sur une carte. Une telle application peut être intéressante pour les étudiants à la recherche d'un stage, ou simplement pour encourager la collaboration entre chercheurs. Elle est restreinte à la base de données *Pubmed* du NCBI, car en plus d'être la base de données que nous connaissons le mieux, elle présente aussi l'avantage de permettre de formaliser la notion de sujet de recherche : un sujet est ramené à une requête *Pubmed*.

L'affichage de la carte se fait côté client. Une requête *Pubmed* est entrée par l'utilisateur, et les données en résultant sont traitées pour déterminer les instituts les plus représentés. Un des problèmes à ce niveau est que le NCBI ne structure pas les informations sur les instituts, nous avons donc été obligés de les « parser ». Nous récupérons les informations intéressantes grâce à une liste de mots-clés (université, institut...) [1]. Le pays est récupéré en essayant de « matcher » une liste exhaustive des pays du monde, et cela marche plutôt bien. Cependant cette approche n'est pas possible avec les villes. Nous avons essayé. Le processus est bien trop long (plus de 96000 villes dans le monde), et trop aléatoire car certaines villes se nommant « as », ou « es » par exemple, l'expression régulière peut « matcher » n'importe quoi.

Après cela, ces données sont envoyées au serveur pour constituer une base de données que les utilisateurs feront évoluer avec leurs requêtes (nous n'allons pas cloner la base du données *Pubmed*). Le contenu de la base de données est alors utilisé afin d'afficher des informations telles que :

— Quel institut conviendrait à un utilisateur donné ?
— Quel sont les collaborateurs idéaux d'un utilisateur ?
— Quel est le champ d'expertise d'un utilisateur ?
— Quel est le « meilleur » institut ? (sur la base des publication et du facteur d'impact)

1. à ce niveau nous sommes contentés de l'Anglais, du Français, de l'Allemand, de l'Italien, de l'Espagnol et du Portugais

— Quel est le « meilleur » chercheur ? (sur la base des publication et du facteur d'impact)
— Quel est la représentativité de chaque pays présent dans la base ?
— Le nombre d'articles en fonction du temps.

2 Création de la base de donnée

Nous avons choisi de créer 8 tables principales pour stocker les informations concernant :
— les utilisateurs : `user` ;
— les articles : `article` ;
— les mots-clé : `keyword` ;
— les qualificateurs : `qualifier` ;
— les requêtes *Pubmed* : `query` ;
— les journaux : `journal` ;
— les auteurs : `author` ;
— les instituts : `affiliation` .

À cela s'ajoute 8 tables relationnelles, contenant uniquement des références, et portant toute la redondance de la base de données :
— `has_searched` (un utilisateur fait une recherche) ;
— `results_in` (une recherche aboutit à des articles) ;
— `has_published` (un journal publit des articles) ;
— `has_keyword` (un article a des mots-clés) ;
— `has_qualifier` (un mot-clé a des qualificateurs) ;
— `has_written` (un auteur écrit des articles) ;
— `belongs_to` (un article a été produit dans un/des instituts) ;
— `is_affiliated_to` (un auteur est affilié à un/des instituts).

La diagramme résultant de la création des tables est représenté dans la figure 1. Nous avons écrit le code pour la créer, et le schéma a été généré par *MySQlWorkbench* à partir de ce code.

```
1  -- creation de la base de donnees
   CREATE DATABASE IF NOT EXISTS prwb;
3  USE prwb;

5  -- creation de la table des utilisateurs
   CREATE TABLE IF NOT EXISTS user ( login CHAR(30) NOT NULL, firstname CHAR(30) NOT
       NULL, lastname CHAR(30) NOT NULL, password CHAR(30) NOT NULL, PRIMARY KEY (login)
       ) ENGINE=InnoDB DEFAULT CHARSET=utf8;
7
   -- creation de la table  des articles
9  CREATE TABLE IF NOT EXISTS article ( pmid INT(10) NOT NULL, title CHAR(200), pubdate
       DATE, PRIMARY KEY (pmid) ) ENGINE=InnoDB DEFAULT CHARSET=utf8;

11 -- creation de la table des mots-cles
   CREATE TABLE IF NOT EXISTS keyword ( value CHAR(100) NOT NULL, PRIMARY KEY (value) )
       ENGINE=InnoDB DEFAULT CHARSET=utf8;
13
   -- creation de la table des qualificateurs
15 CREATE TABLE IF NOT EXISTS qualifier ( value CHAR(40) NOT NULL, PRIMARY KEY (value) )
        ENGINE=InnoDB DEFAULT CHARSET=utf8;

17 -- creation de la table des auteurs
   CREATE TABLE IF NOT EXISTS author ( id INT(5) NOT NULL AUTO_INCREMENT, lastname CHAR
       (40) NOT NULL, firstname CHAR(40), initials CHAR(10), PRIMARY KEY (id), UNIQUE KEY
        id_author (lastname, firstname, initials) ) ENGINE=InnoDB DEFAULT CHARSET=utf8;
19
   -- creation de la table requetes
```

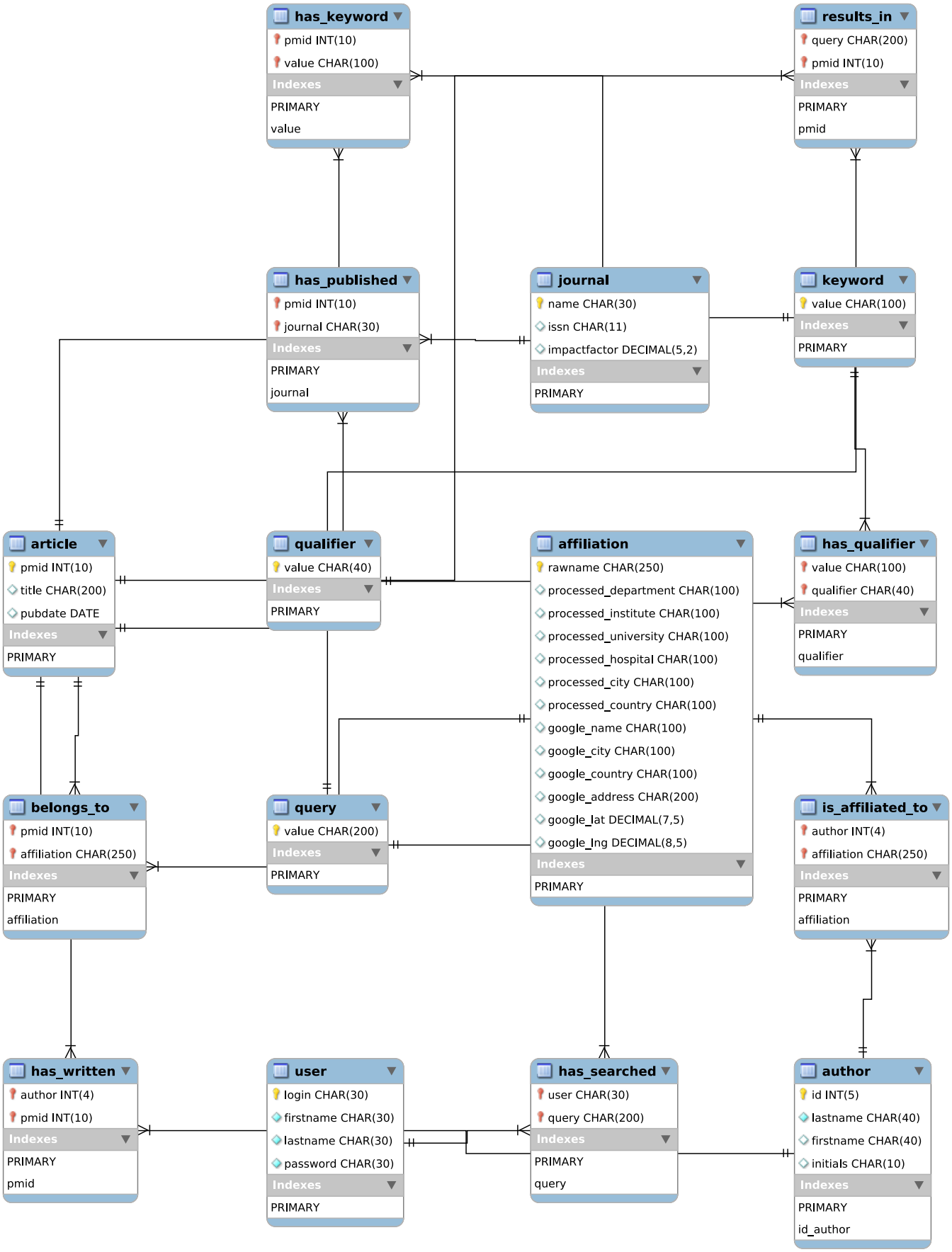

FIGURE 1 – Diagramme du modèle de relations entre entités.

```
21 CREATE TABLE IF NOT EXISTS query ( value CHAR(200) NOT NULL, PRIMARY KEY (value) )
     ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

```sql
-- creation de la table des journaux
CREATE TABLE IF NOT EXISTS journal ( name CHAR(30) NOT NULL, issn CHAR(11),
    impactfactor DECIMAL(5,2), PRIMARY KEY (name) ) ENGINE=InnoDB DEFAULT CHARSET=utf8
    ;

-- creation de la table des affiliations
CREATE TABLE IF NOT EXISTS affiliation ( rawname CHAR(250) NOT NULL,
    processed_department CHAR(100), processed_institute CHAR(100),
    processed_university CHAR(100), processed_hospital CHAR(100), processed_city CHAR
    (100), processed_country CHAR(100), google_name CHAR(100), google_city CHAR(100),
    google_country CHAR(100), google_address CHAR(200), google_lat DECIMAL(7,5),
    google_lng DECIMAL(8,5), PRIMARY KEY (rawname) ) ENGINE=InnoDB DEFAULT CHARSET=
    utf8;

-- creation de la table relationnelle : un article a pour mot(s)-cle(s)
CREATE TABLE has_keyword ( pmid INT(10), value CHAR(100), FOREIGN KEY (pmid)
    REFERENCES article(pmid), FOREIGN KEY (value) REFERENCES keyword(value), PRIMARY
    KEY (pmid, value) ) ENGINE=InnoDB DEFAULT CHARSET=utf8;

-- creation de la table relationnelle : un mot-cle a pour qualificateur(s)
CREATE TABLE has_qualifier ( value CHAR(100), qualifier CHAR(40), FOREIGN KEY (value)
     REFERENCES keyword(value), FOREIGN KEY (qualifier) REFERENCES qualifier(value),
    PRIMARY KEY (value, qualifier) ) ENGINE=InnoDB DEFAULT CHARSET=utf8;

-- creation de la table relationnelle : un auteur a ecrit un (des) article(s)
CREATE TABLE has_written ( author INT(4), pmid INT(10), FOREIGN KEY (author)
    REFERENCES author(id), FOREIGN KEY (pmid) REFERENCES article(pmid), PRIMARY KEY (
    author, pmid) ) ENGINE=InnoDB DEFAULT CHARSET=utf8;

-- creation de la table relationnelle : un journal a edite un( des) article(s)
CREATE TABLE has_published ( pmid INT(10), journal CHAR(30), FOREIGN KEY (journal)
    REFERENCES journal(name), FOREIGN KEY (pmid) REFERENCES article(pmid), PRIMARY KEY
     (pmid, journal) ) ENGINE=InnoDB DEFAULT CHARSET=utf8;

-- creation de la table relationnelle : un auteur a une (des) affiliation(s)
CREATE TABLE is_affiliated_to ( author INT(4), affiliation CHAR(250), FOREIGN KEY (
    author) REFERENCES author(id), FOREIGN KEY (affiliation) REFERENCES affiliation(
    rawname), PRIMARY KEY (author, affiliation) ) ENGINE=InnoDB DEFAULT CHARSET=utf8;

-- creation de la table relationnelle : un article provient de un (des) affiliation(s
    )
CREATE TABLE belongs_to ( pmid INT(10), affiliation CHAR(250), FOREIGN KEY (pmid)
    REFERENCES article(pmid), FOREIGN KEY (affiliation) REFERENCES affiliation(rawname
    ), PRIMARY KEY (pmid, affiliation) ) ENGINE=InnoDB DEFAULT CHARSET=utf8;

-- creation de la table relationnelle : une requete donne zero, un ou des article(s)
CREATE TABLE results_in ( query CHAR(200), pmid INT(10), FOREIGN KEY (query)
    REFERENCES query(value), FOREIGN KEY (pmid) REFERENCES article(pmid), PRIMARY KEY
    (query, pmid) ) ENGINE=InnoDB DEFAULT CHARSET=utf8;

-- creation de la table relationnelle : un utilisateur a effectue une (des) requete(s
    )
CREATE TABLE has_searched ( user CHAR(30), query CHAR(200), FOREIGN KEY (user)
    REFERENCES user(login), FOREIGN KEY (query) REFERENCES query(value), PRIMARY KEY (
    user, query) ) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

Listing 1 – Code SQL pour la création de la base de données.

3 Requêtes

3.1 Insertion

Voici une liste exhaustive des requêtes que nous utilisons pour entrer des informations dans la base de données. Cela se fait à la volée au fur et à mesure des requêtes des utilisateurs.

```
-- enregistrement d'un utilisateur
INSERT INTO user (login, lastname, firstname, password) VALUES ("jpmarielle","marielle","jean-pierre","coupdetorchon");

-- quelle a ete la recherche ?
INSERT IGNORE INTO query (value) VALUES ('synthetic[All Fields] AND ("chromosomes"[MeSH Terms] OR "chromosomes"[All Fields] OR "chromosome"[All Fields])');

-- quel utilisateur a effectue cette recherche ?
INSERT IGNORE INTO has_searched (user, query) VALUES ("jpmarielle","breast cancer");

-- a quel(s) articles a abouti la recherche ?
INSERT IGNORE INTO results_in (query, pmid) VALUES ('synthetic[All Fields] AND ("chromosomes"[MeSH Terms] OR "chromosomes"[All Fields] OR "chromosome"[All Fields])','24674868");

-- quel(s) est (sont) cet (ces) articles ?
INSERT IGNORE INTO article (pmid, title, pubdate) VALUES ("24674868","Total synthesis of a functional designer eukaryotic chromosome.", "2014");

-- dans quel journal ?
INSERT IGNORE INTO has_published (pmid, journal) VALUES ("24674868","Science");

-- on ajoute le journal
INSERT IGNORE INTO journal (name, issn, impactfactor) VALUES ("Science","0036-8075","31.48");

-- quels sont ses mots-cles ?
INSERT IGNORE INTO has_keyword (pmid, value) VALUES ("24674868","Chromosomes, Fungal");

-- on ajoute les mots-cle
INSERT IGNORE INTO keyword (value) VALUES ("Chromosomes, Fungal");

-- on ajoute les qualificateurs
INSERT IGNORE INTO qualifier (value) VALUES ("genetics");

-- on ajoute les qualificateurs correspondant aux mots-cles
INSERT IGNORE INTO has_qualifier (value, qualifier) VALUES ("Chromosomes, Fungal","genetics");

-- qui a ecrit cet article ?
INSERT IGNORE INTO has_written (author, pmid) VALUES (id,"24674868");

-- on ajoute l'auteur
INSERT IGNORE INTO author (lastname, firstname, initials) VALUES ("Annaluru", "Narayana", "N");

-- par quel institut a-t-il ete publie ?
INSERT IGNORE INTO belongs_to (pmid, affiliation) VALUES ("24674868", "Department of Environmental Health Sciences, Johns Hopkins University (JHU) School of Public Health, Baltimore, MD 21205, USA");

-- on ajoute l'institut
INSERT IGNORE INTO affiliation (rawname, processed_department, processed_institute, processed_university, processed_hospital, processed_city, processed_country,
```

```
    google_name, google_city, google_country, google_address, google_lat, google_lng)
    VALUES ("Department of Biochemistry, Hong Kong University of Science and
    Technology, Clear Water Bay, Hong Kong","department of biochemistry","","hong kong
     university of science and technology","","","hong kong","Hong Kong University of
    Science and Technology","Hong Kong","Hong Kong University of Science and
    Technology, Clear Water Bay, Hong Kong","22.33640","114.26547");

-- a quel institut est affilie l'auteur ?
INSERT IGNORE INTO is_affiliated_to (author, affiliation) VALUES (id,"Department of
    Environmental Health Sciences, Johns Hopkins University (JHU) School of Public
    Health, Baltimore, MD 21205, USA");
```

Listing 2 – Code SQL pour l'insertion d'information dans la base de données.

3.2 Interrogation

Voici une liste exhaustive des requêtes que nous utilisons pour interroger la base de données.

```
-- les sujets favoris de l'utilisateur
SELECT @rank := @rank + 1 AS rank, results.* FROM ( SELECT user.login, has_qualifier.
    value, COUNT(has_qualifier.value) AS occurence FROM user INNER JOIN has_searched
    INNER JOIN results_in INNER JOIN has_keyword INNER JOIN has_qualifier ON ( user.
    login = "jpmarielle" AND user.login = has_searched.user AND has_searched.query =
    results_in.query AND results_in.pmid = has_keyword.pmid AND has_keyword.value =
    has_qualifier.value ) GROUP BY has_qualifier.qualifier ORDER BY COUNT(*) DESC
    LIMIT 10) results CROSS JOIN (SELECT @rank := 0) init;

-- les mots-cles favoris de l'utilisateur
SELECT @rank := @rank + 1 AS rank, results.* FROM ( SELECT user.login, has_keyword.
    value, COUNT(has_keyword.value) AS occurence FROM user INNER JOIN has_searched
    INNER JOIN results_in INNER JOIN has_keyword ON ( user.login = "jpmarielle" AND
    user.login = has_searched.user AND has_searched.query = results_in.query AND
    results_in.pmid = has_keyword.pmid ) GROUP BY has_keyword.value ORDER BY COUNT(*)
    DESC LIMIT 10) results CROSS JOIN (SELECT @rank := 0) init;

-- l'institut favori de l'utilisateur
SELECT belongs_to.affiliation AS affiliation FROM user INNER JOIN has_searched INNER
    JOIN results_in INNER JOIN belongs_to ON ( user.login = "jpmarielle" AND user.
    login = has_searched.user AND has_searched.query = results_in.query AND results_in
    .pmid = belongs_to.pmid ) GROUP BY belongs_to.affiliation ORDER BY COUNT(*) DESC
    LIMIT 3;

-- les chercheurs favoris de l'utilisateur
SELECT author.firstname, author.lastname FROM user INNER JOIN has_searched INNER JOIN
     results_in INNER JOIN has_written INNER JOIN author ON ( user.login = "jpmarielle
    " AND user.login = has_searched.user AND has_searched.query = results_in.query AND
     results_in.pmid = has_written.pmid AND has_written.author = author.id) GROUP BY
    author.id ORDER BY COUNT(*) DESC LIMIT 5;

-- les recherches de l'utilisateur sont-elles a la mode ?
SELECT EXTRACT(YEAR FROM article.pubdate) year, COUNT(EXTRACT(YEAR FROM article.
    pubdate)) AS hit FROM user INNER JOIN has_searched INNER JOIN results_in INNER
    JOIN article ON ( user.login = "jpmarielle" AND user.login = has_searched.user AND
     has_searched.query = results_in.query AND results_in.pmid = article.pmid AND
    EXTRACT(YEAR FROM article.pubdate) IS NOT NULL ) GROUP BY year ORDER BY year;

-- classement des instituts
SELECT affiliation.rawname, AVG(journal.impactfactor) AS score FROM affiliation INNER
     JOIN belongs_to INNER JOIN article INNER JOIN has_published JOIN journal ON (
    affiliation.rawname = belongs_to.affiliation AND belongs_to.pmid = article.pmid
    AND article.pmid = has_published.pmid AND has_published.journal = journal.name)
    GROUP BY affiliation.rawname ORDER BY score DESC;
```

```sql

-- classement des auteurs
SELECT author.firstname, author.lastname, is_affiliated_to.affiliation, AVG(journal.impactfactor) AS score FROM author INNER JOIN is_affiliated_to INNER JOIN has_written INNER JOIN article INNER JOIN has_published JOIN journal ON (author.id = is_affiliated_to.author AND author.id = has_written.author AND has_written.pmid = article.pmid AND article.pmid = has_published.pmid AND has_published.journal = journal.name) GROUP BY author.firstname, author.lastname ORDER BY score DESC;

-- representativite des pays
SELECT processed_country, ( COUNT(processed_country) * 100 / (SELECT COUNT(*) FROM affiliation) ) AS frequency FROM affiliation WHERE affiliation.processed_country != '' GROUP BY affiliation.processed_country ORDER BY frequency DESC;

-- evolution des publications au cours du temps
SELECT EXTRACT(YEAR FROM article.pubdate) year, COUNT( EXTRACT(YEAR FROM article.pubdate) ) AS hit FROM article WHERE EXTRACT(YEAR FROM article.pubdate) IS NOT NULL GROUP BY year ORDER BY year;

-- quel est l'identifiant correspondant a ce nom ? (necessaire car la cle est l'identifiant)
SELECT id FROM author WHERE lastname = "Annaluru" AND firstname = "Narayana" AND initials = "N";

-- ce login est-il disponible ? (necessaire a l'inscription)
SELECT login FROM user WHERE login = "jpmarielle";

-- quelles sont les informations correspondant a ce login ? (necessaire a l'idenfification)
SELECT * FROM user WHERE login = "jpmarielle";
```

Listing 3 – Code SQL pour l'interrogation de la base de données.

4 Informations complémentaires

Nous nous excusons pour la propreté du code javascript, qui n'est pas du tout orienté objet. Cela n'est pourtant pas faute d'avoir essayé, cependant la maîtrise des objets différés était nécessaire, et nous avons eu des difficultés, non pas pour les utiliser, mais pour les intégrer aux fonctions [2].

L'aspect visuel du site a été repris sur le site de boostrap, pour la navbar notamment, et notre formulaire de connection est fortement inspiré de ces deux sites :

- `http://www.bootply.com/60886`
- `http://mrbool.com/how-to-create-a-sign-up-form-registration-with-php-and-mysql/28675`

Le code pour l'affichage du nom de l'institut sur les marqueurs de la carte a été pris ici :

- `http://gmap3.net/en/catalog/18-data-types/event-65`

Autrement, et sauf si spécifié dans le code, tout le reste du code vient de nous (avec l'aide de *stackoverflow*).

Les listes des pays et des villes du monde ont été récupérés sur ces sites :

- `http://www.unece.org/cefact/codesfortrade/codes_index.html`

2. ce n'est pourtant pas compliqué *a priori*, mais nous avons eu des erreurs systématiques à ce niveau

— `https://commondatastorage.googleapis.com/ckannet-storage/2011-11-25T132653/iso_3166_2_countries.csv`

Pour finir nous attirons votre attention sur trois points :

— Le site marche pour la plupart des requêtes que nous avons testé, cependant nous sommes limités en nombre par les différentes API que nous utilisons. Il n'y a donc pas de problème avec les requêtes donnant quelques centaines de résultats, mais il peut y en avoir avec les requêtes donnant un nombre de résultat de l'ordre du millier. Vous pouvez tester ces trois requêtes à titre d'exemple pour commencer :
  — `reed survivin` (24 résultats) ;
  — `cdc20 review` (73 résultats) ;
  — `apc cdc20 mad2` (123 résultats).

— Nous avons du mal à faire correspondre les résultats de *googlemap* avec nos requêtes. Nous utilisons une expression régulière pour cela, et cela marche plus ou moins bien. Cela n'empêche pas les résultats de s'afficher sur la carte, cependant il est fréquent que les informations issues de la géolocalisation ne soient pas intégrées dans la base de données.

— Nous devions intégrer une protection contre l'injection SQL, avec une utilisation de `?` et d'une `array` pour faire des requêtes en deux temps. Cependant cela n'a pas été intégré car nous venons de nous en rendre compte. Nous nous excusons pour cela.

Le login que vous pouvez utiliser pour tester le site est `jpmarielle` avec le mot de passe `coupdetorchon`. Le « dump » de la base de données a été effectué avec `mysqldump` [3] et sa localisation est `/sql/database.sql`. La base de données a été remplie à l'aide de quelques requêtes car elle est susceptible de grossir rapidement.

Merci beaucoup pour le temps passé à lire ce rapport.

3. nous espérons que cela vous convient