BDBIO - Exemple de projet

Fabien Duchateau

fabien.duchateau [at] univ-lyon1.fr

Université Claude Bernard Lyon 1

2023 - 2024

https://perso.liris.cnrs.fr/fabien.duchateau/BDBIO/

Informations générales

Page git du projet :

- https://forge.univ-lyon1.fr/fabien.duchateau/bdbio-projet
- description du sujet (`projet.md`)
- suivi du projet (`dashboard.md`)
- modèle de rapport (`rapport.md`)

Page git des adaptateurs (wrappers) :

- https://forge.univ-lyon1.fr/fabien.duchateau/bdbio-wrappers
- interactions simplifiées pour MongoDB, PosgreSQL, fichier XML, BlazeGraph (et d'autres)
- installation :

```
python3 -m pip install git+https://fduchate@forge.univ-lyon1.fr/
fabien.duchateau/bdbio-wrappers.git
```

Source de données

Nirvana live performances :

- format CSV
- 6 attributs
- 415 instances

```
??/??/1987, house party, Aberdeen, WA, United States, N
27/06/1987, Community World Theater, WA, Tacoma, United States, N
30/08/1992, Richfield Avenue (Reading Festival),, Reading, United Kingdom, Y
...
```

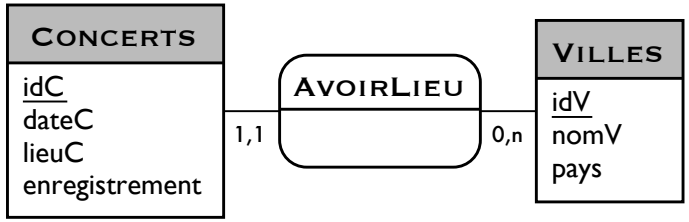
Pour simplifier l'exemple, l'état (e.g., WA) n'est pas conservé lors de la migration

https://data.world/ben-pfeifer/nirvana-live-performances

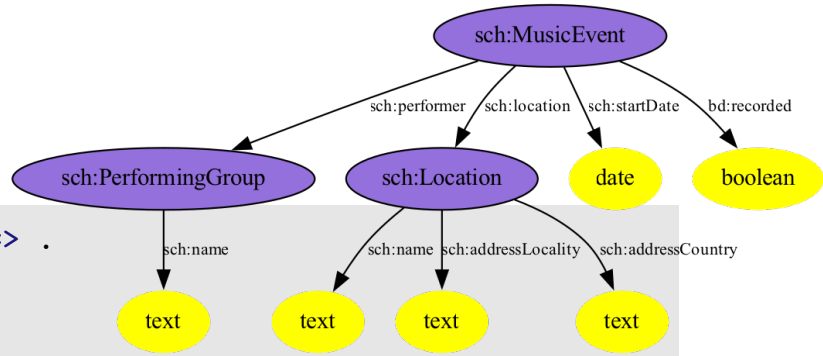
Migration - modélisation

Objectif : migrer la source de données `nirvana.csv` en graphe

- Diagramme entité-association



- Schéma logique



- Triplets RDF de description

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix sch: <http://schema.org/> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix bd: <http://projet.bd/> .

sch:performer rdfs:domain sch:MusicEvent .
sch:performer rdfs:range sch:PerformingGroup .
sch:startDate rdfs:domain sch:MusicEvent .
sch:startDate rdfs:range sch:Date .
bd:recorded rdfs:domain sch:MusicEvent .
bd:recorded rdfs:range sch:Boolean .
sch:location rdfs:domain sch:MusicEvent .
sch:location rdfs:range sch:Location .
```

Migration - code (1/2)

Problèmes pour la migration :

- Transformation des valeurs *N/Y* en *false/true*
- Certaines dates (e.g., *??/??/1987*) ne sont pas valides pour un type `DATE`

```
1  import csv
2  import os
3
4  data_csv = os.path.join('data', 'Nirvana Performances.csv')
5  lieux = dict()  # {lieu_ville_pays: URI}, pour vérifier si lieux déjà existants
6  compteur_concerts = compteur_lieux = 1
7
8  if __name__ == "__main__":
9      print(buffer_preambule)
10     with open(data_csv, newline='') as csv_file:  # lecture du CSV
11         reader = csv.reader(csv_file, delimiter=',', quotechar='"')
12         try:
13             next(reader, None)  # passer l'entête du CSV
14             for row in reader:  #  date, lieu, ville, [état], pays, enregistrement
15                 buffer_concert = ""
```

Début de l'algorithme (imports, parcours du fichier csv)

Migration - code (2/2)

```
16                uri = "bd:C" + str(compteur_concerts)
17                buffer_concert += f"{uri} rdf:type sch:MusicEvent .\n"
18                buffer_concert += f"{uri} sch:performer bd:Nirvana .\n"
19                buffer_concert += f"{uri} sch:startDate \"{row[0]}\" .\n"
20                enregistrement = "true" if row[5] == 'Y' else "false"  # transformation chaine
                      en booléen
21                buffer_concert += f"{uri} bd:recorded {enregistrement} .\n"
22                cle_lieu = f"{row[1]}_{row[2]}_{row[4]}"
23                if cle_lieu in lieux:  # lieu existant, récupération de son URI
24                    uri_lieu = lieux[cle_lieu]
25                else:  # création d'une URI pour le lieu
26                    uri_lieu = "bd:L" + str(compteur_lieux)
27                    buffer_concert += f"{uri_lieu} sch:name \"{row[1]}\" .\n"
28                    buffer_concert += f"{uri_lieu} sch:addressLocality \"{row[2]}\" .\n"
29                    buffer_concert += f"{uri_lieu} sch:addressCountry \"{row[4]}\" .\n"
30                    lieux[cle_lieu] = uri_lieu
31                    compteur_lieux += 1
32                buffer_concert += f"{uri} sch:location {uri_lieu} ."
33                compteur_concerts += 1
34                print(buffer_concert)
35            except csv.Error as e:
36                print(f"Erreur de parsing ({data_csv} - {reader.line_num}) : {e}")
37            print(f"Terminé ({compteur_concerts} concerts, {compteur_lieux} lieux)")
```

Suite et fin (génération des triplets, avec vérification de l'existence d'un lieu via un dictionnaire)

Migration - résultat et pseudo-code

```
...  # derniers triplets générés
bd:C415 rdf:type sch:MusicEvent .
bd:C415 sch:performer bd:Nirvana .
bd:C415 sch:startDate "01/03/1994" .
bd:C415 bd:recorded true .
bd:L350 sch:name "Terminal 1, Flughafen München-Riem" .
bd:L350 sch:addressLocality "Munich" .
bd:L350 sch:addressCountry "Germany" .
bd:C415 sch:location bd:L350 .
```

Résultat : 415 concerts, 350 lieux (3130 triplets)

```
1  print(preambule)  # graphe de description (préfixes, schéma)
2  pour chaque ligne de 'Nirvana performances.csv':
3    uri = 'C' + compteur_uri++
4    # construction des triplets selon le schéma (exemples, idem pour autres prédicats)
5    print(uri + " sch:startDate " + row['date'] + " .")  # triplet date
6    enregistrement = true si 'Y' sinon false  # transformation valeur N/Y en true/false
7    print(uri + " bd:recorded " + enregistrement + " .")  # triplet enregistrement
8    clé_lieu = row['lieu'] + row['ville'] + row['pays']
9    si clé_lieu in lieux:  # lieu existant
10     uri_lieu = lieux[clé_lieu]
11   sinon:  # création nouveau lieu
12     uri_lieu = 'L' + compteur_lieu++
13     print(uri_lieu + " sch:name " + row['lieu'] + " .")  # triplet nom lieu, idem ville/pays
14     lieux[clé_lieu] = uri_lieu  # ajout du lieu dans lieux existants
```

Exemple d'algorithme en pseudo code pour le rapport

À vous de jouer !

La suite (partie interrogation distribuée) à 13h30...

Des questions ?

Le chat (Geluck, https://lechat.com/)

Interrogation distribuée - jeux de données

Objectif (question) : le groupe Nirvana a-t-il joué dans des villes de plus en plus grandes au fil des années ?

Utilisation d'un autre jeu de données :

- Un document représente un quartier d'une ville (aux US)
- 29543 documents, 5 propriétés
- Stockage dans une collection `zips`


```
{ "_id" : "01001", "city" : "AGAWAM", "loc" : [
    -72.622739, 42.070206 ], "pop" : 15338, "state" :
    "MA" }
{ "_id" : "01002", "city" : "CUSHMAN", "loc" : [
    -72.51564999999999, 42.377017 ], "pop" : 36963,
    "state" : "MA" }
{ "_id" : "57465", "city" : "NORTHVILLE", "loc" : [
    -98.65885299999999, 45.161112 ], "pop" : 328,
    "state" : "SD" }
...
```

```
mif04> db.zips.findOne();
{
  _id: '01001',
  city: 'AGAWAM',
  loc: [ -72.622739, 42.070206 ],
  pop: 15338,
  state: 'MA'
}
```

https://media.mongodb.org/zips.json

Interrogation distribuée - problèmes

Contrainte : seulement les concerts aux USA

Problèmes pour l'intégration :

- Casse différente sur les noms de ville
- Dans `zips`, même nom de ville dans différents états (e.g., Aberdeen)
- Quelques villes sans correspondance (e.g., typo, autre forme)

```
mif04> db.zips.aggregate(
...     { $match: {city: "ABERDEEN"}},
...     { "$group": {
.....         "_id": { city: "$city", state: "$state" },
.....         "total": { $sum: "$pop" },
.....     }},
...     { $sort: { "total": -1 }},
...     { $limit: 10 },);
[
  { _id: { city: 'ABERDEEN', state: 'SD' }, total: 28786 },
  { _id: { city: 'ABERDEEN', state: 'WA' }, total: 22346 },
  { _id: { city: 'ABERDEEN', state: 'MD' }, total: 19229 },
  { _id: { city: 'ABERDEEN', state: 'MS' }, total: 15769 },
  { _id: { city: 'ABERDEEN', state: 'NC' }, total: 7767 },
  { _id: { city: 'ABERDEEN', state: 'OH' }, total: 2176 }
]
```

```
mif04> db.zips.find({"city": {$regex: /BAINBRIDGE/}});
[
  {
    _id: '13733',
    city: 'BAINBRIDGE',
    loc: [ -75.489411, 42.311975 ],
    pop: 5073,
    state: 'NY'
  },
  {
    _id: '17502',
    city: 'BAINBRIDGE',
    loc: [ -76.672589, 40.1086 ],
    pop: 2688,
    state: 'PA'
  },
  {
    _id: '21904',
    city: 'BAINBRIDGE',
    loc: [ -76.083706, 39.622264 ],
    pop: 5507,
    state: 'MD'
  },
  {
    _id: '31717',
    city: 'BAINBRIDGE',
    loc: [ -84.573975, 30.897865 ],
    pop: 17739,
    state: 'GA'
  },
  {
    _id: '45612',
    city: 'BAINBRIDGE',
    loc: [ -83.276268, 39.213116 ],
    pop: 4356,
    state: 'OH'
  },
  {
    _id: '46105',
    city: 'BAINBRIDGE',
    loc: [ -86.771119, 39.740664 ],
    pop: 3147,
    state: 'IN'
  },
  {
    _id: '98110',
    city: 'BAINBRIDGE ISLAN',
    loc: [ -122.531297, 47.645048 ],
    pop: 15846,
    state: 'WA'
  }
```

Interrogation distribuée - code (1/2)

```
1  from wrappers.mongo import WrapperMongo
2  from wrappers.blazegraph import WrapperBlazegraph
3
4  if __name__ == "__main__":
5      # connexion à la BD nirvana et la BD zips
6      blazegraph =
           WrapperBlazegraph(server='http://192.168.77.137:8080/blazegraph/namespace/nirvana')
7      mongo = WrapperMongo(server="mongodb://bd-pedago.univ-lyon1.fr:27017/mif04",
           username="mif04", password=passd, db="mif04", collection="zips")
8      results = {}
9      # pour chaque concert aux US, récupère la date et ville
10     req = """prefix sch:<http://schema.org/>
11     select ?date ?ville
12     where {?e sch:location ?l .
13        ?e sch:startDate ?date .
14        ?l sch:addressLocality ?ville .
15        ?l sch:addressCountry "United States" .
16     }"""
17     concerts = blazegraph.query_sparql(req)
```

Début de l'algorithme (import, connexion aux BD, récupération des concerts)

```
corresp_villes = {"BAINBRIDGE ISLAND": "BAINBRIDGE ISLAN", "PITTSBURGH": "EAST PITTSBURGH",
    "ST. LOUIS": "SAINT LOUIS", "ST. PAUL": "SAINT PAUL"}
```

Une solution pour les villes inconnues : une table de correspondances

Interrogation distribuée - code (2/2)

```
18      for concert in concerts['results']['bindings']:  # structure résultat blazegraph
19          annee = int(concert['date']['value'][-4:])  # extraction de l'année à partir de la date
20          nomv = str(concert['ville']['value']).upper()  # besoin de mettre la ville en
                majuscules
21          if nomv in corresp_villes:  # si ville spéciale, on remplace son nom
22              nomv = corresp_villes[nomv]
23          # même ville dans différents états - ici choix de la plus grande ville
24          villes = mongo.aggregate([{"$match": {"city": nomv}},
25                                    {"$group": {
26                                        "_id": {"city": "$city", "state": "$state"},
27                                        "total": {"$sum": "$pop"},
28                                    }},
29                                    {"$sort": {"total": -1}},
30                                    {"$limit": 1}])
31          # mise à jour du dict de comptage
32          if annee not in results:
33              results[annee] = {"sommepop": 0, "nbvilles": 0}
34          results[annee]["sommepop"] += int(list(villes)[0]['total'])  # on somme la population
                de la ville
35          results[annee]["nbvilles"] += 1  # une ville supplémentaire pour l'année
36      # affichage des résultats (année : moyenne de population des villes)
37      for annee, stats in results.items():
38          moypop = int(stats['sommepop'] / stats['nbvilles'])
39          print(f"{annee} : {moypop} habitants en moyenne ({stats['nbvilles']} villes)")
```

Suite et fin de l'algorithme (pour chaque concert, récupération de la population agrégée de la plus grande ville et affichage final)

Interrogation distribuée - pseudo-code

```
bdbg = connexion('nirvana')  # connexion à la BD nirvana
bdmongo = connexion('zips')  # connexion à la BD zips
concerts = bdbg.executer("""prefix sch:<http://schema.org/>
    select ?date ?ville
    where {?e sch:location ?l .
       ?e sch:startDate ?date .
       ?l sch:addressLocality ?ville .
       ?l sch:addressCountry "United States" .
    }""")
results = {}
for concert in concerts:
    annee = concert['date'][-4:]  # extraction année
    nomv = concert['ville'].upper()  # ville en majuscule
    si nomv dans tab_correspondances:  # la ville a un nom alternatif
        nomv = tab_correspondances[nomv]  # mise à jour du nom
    ville = bdmongo.executer([{"$match": {"city": nomv}},
                              {"$group": {
                                   "_id": {"city": "$city", "state": "$state"},
                                   "total": {"$sum": "$pop"},
                               }},
                               {"$sort": {"total": -1}},
                               {"$limit": 1}])
    results[annee] = {pop += ville['total'], nbv += 1}  # somme population, incrément nb ville
for année in results:  # calcul de la moyenne de population par année
    moy = results[annee][pop] / results[annee][nbv]
    print(annee : moy)
```

Exemple de pseudo code pour le rapport

À vous de jouer !

```
1987 : 110923 habitants en moyenne (total de 7 villes)
1988 : 288247 habitants en moyenne (total de 23 villes)
1989 : 605565 habitants en moyenne (total de 45 villes)
1990 : 476852 habitants en moyenne (total de 52 villes)
1991 : 493143 habitants en moyenne (total de 58 villes)
1992 : 972999 habitants en moyenne (total de 18 villes)
1993 : 685968 habitants en moyenne (total de 53 villes)
1994 : 325091 habitants en moyenne (total de 5 villes)}
```

Résultat : oui, le groupe a joué dans des villes de plus en plus grandes (avec exceptions certaines années, e.g., tournée hors US)

Des questions ?