[MIF14] Bases de données déductives

CHASSIN DE KERGOMMEAUX Loïc, MERCIER Loris

Juin 2023

Rapport de projet :
Fonction d'agrégation pour Datalog

---

1 Introduction

Dans le cadre de l'UE *MIF14 : Bases de données déductives,* nous avons été amenés à implémenter un moteur de requête Datalog prenant en compte des fonctions d'agrégation telles que COUNT, AVG, SUM, etc… Pour ce faire, nous avons choisi d'utiliser le langage de programmation *Python* pour sa flexibilité et sa présence de bibliothèques performantes pour le traitement de données. Nous reviendrons d'ailleurs dans ce rapport sur le fonctionnement général de notre implémentation (Parsing puis évaluation) avant d'analyser quelques uns de nos résultats.

2 Lancement de l'application

Notre module Datalog s'appuie sur un système de lecture/écriture de fichier en qualité d'input/output. L'utilisateur doit ainsi écrire son programme Datalog dans un fichier *.txt,* respectant la syntaxe usuelle du langage, afin que notre application puisse en lire et en évaluer ses règles.

Lors de l'appel à notre module, l'utilisateur doit ensuite spécifier le nom de ses fichiers en argument suivant cette commande : *py main.py FILENAME_IN FILENAME_OUT.txt.* Notre application lancera alors l'évaluation du fichier *FILENAME_IN* avant d'en écrire le résultat dans le fichier *FILENAME_OUT.txt.* Si *FILENAME_OUT* n'est pas défini, le module créera un fichier out.txt pour écrire les résultats calculés.

3 Notre parseur

3.1 Règles de validité d'un programme

Notre parseur vérifie les règles suivantes :

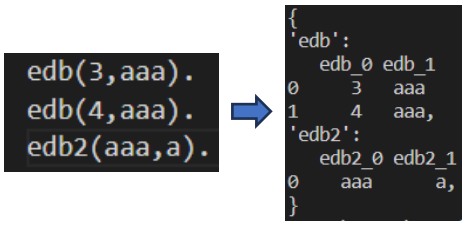
- Une ligne d'un programme est soit un commentaire, soit un EDB, soit un IDB.
  - Un commentaire doit débuter par un '#' suivi d'un espace.
  - Les EDB et IDB doivent démarrer avec une lettre minuscule et finir par un point.
  - Deux EDB du même nom doivent posséder un nombre d'argument identique.
  - Un IDB est caractérisé par la présence du symbole ' :-' entre sa tête et son corps.
  - Les prédicats présents dans le corps d'un IDB doivent être séparés d'une virgule.
  - Les fonctions d'agrégation prisent en charge sont COUNT, AVG, SUM, MIN, MAX. Elles possèdent deux arguments (X,Y) où X est la colonne à calculer est Y la valeur résultante.
  - Les opérateurs numériques '==', '!=', '>', '<', '>=', '<=', ne doivent pas contenir d'espace entre leurs arguments. Ex : 'X==Y' est conforme. 'X == Y' n'est pas conforme.
  - Les espaces (autres que pour les opérateurs numériques) sont facultatifs.
- Le caractère '_' indiquant qu'une variable n'a aucune importance est accepté.
- Les variables sont écrites en MASJUSCULES tandis que les atomes sont en minuscules.

Pour chaque erreur soulevée par notre parseur, une exception '*DatalogInternalError'* sera levée indiquant à l'utilisateur la raison et la localisation précise de l'erreur dans son fichier source.

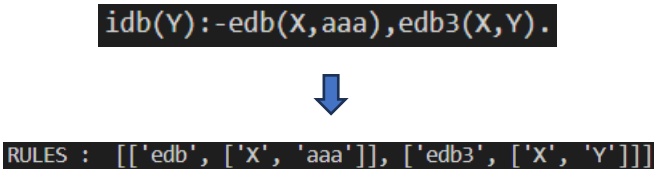
3.2 Formatage des données

Après la validité du programme vérifiée, notre module convertit les règles du programme dans des structures de données propres à nos choix d'implémentation. Concrètement, nous décidons de stocker l'ensemble de nos prédicats dans un dictionnaire. La clé sera le nom du prédicat tandis que sa valeur sera un Dataframe contenant tous les faits du prédicat.

Dans le cas d'un EDB, nous formattons directement la ligne dans le format présenté ci-dessus. Dans le cas d'un IDB, nous devons d'abord évaluer la règle avant de l'intégrer à notre dictionnaire général. La mission du parseur est alors uniquement de convertir le corps d'un IDB en une liste *RULES* de prédicats dont le premier argument contient le nom du prédicat et le second ses arguments. Voici un exemple de parsing ci-dessous :



Exemple de parsing de règles EBD



Exemple de parsing d'une règle IDB avant évaluation

4 Evaluation d'un programme

4.1 Utilisation du module Pandas DataFrame

Etape la plus importante de ce projet, notre évaluation des règles IDB s'appuient sur la structure de données *pandas DataFrame.* Basés sur la bibliothèque *numpy*, les *Dataframes* sont des tableaux deux dimensions spécialement conçus pour le traitement et l'analyse de données. Ils permettent dans le cadre de notre projet de stocker simplement et efficacement l'ensemble de nos données tout en traitant efficacement les opérations de jointures et les fonctions d'agrégation. Basé sur du C, ce module python nous permet donc de réaliser des requêtes de façon rapide et performante y compris sur des grands jeux de données.

4.2 Fonctionnement générale de notre algorithme.

L'évaluation d'un programme Datalog repose essentiellement sur l'évaluation des règles IDB. Pour ce faire, nous décidons d'évaluer pas à pas chaque prédicat présent dans le corps d'un IDB en itérant sur notre liste RULES conçue lors du parsing (*Voir exemple parsing IDB ci-dessus*).

Pour chaque prédicat, nous vérifions d'abord le type de celui-ci. Si nous détectons un EDB ou IDB déjà présent en base : nous remplissons un *DataFrame* temporaire avec les valeurs de la base pour cet EDB/IDB. Ensuite, on regarde les arguments de cet EDB/IDB. Si l'argument passé est un '_', on ne fait rien. Si l'argument passé n'est ni un '_', ni une variable, nous filtrons les valeurs du *DataFrame* temporaire avec cette valeur. Enfin, nous mettons à jour le *DataFrame* contenant le résultat de la requête (=Celui de l'IDB en cours de d'évaluation) en faisant une jointure entre ce *DataFrame* et le *DataFrame temporaire* que nous venons de remplir. La jointure se fait sur les arguments détectés comme variables, c'est-à-dire démarrant en majuscule. Cela correspond à un procédé d'unification.

Si nous détectons une fonction d'agrégation : nous récupérons la colonne sur laquelle porte la fonction d'agrégation. Ensuite, nous faisons un *DataFrame.groupBy()* sur toutes les colonnes du résultat de la requête sauf cette colonne. Enfin, on applique la fonction d'agrégation sur la bonne colonne pour chaque groupe, et nous mettons à jour le *DataFrame* avec le résultat de la fonction d'agrégation pour chaque groupe.

Sinon, si l'on détecte un opérateur numérique comme '==', '>=', etc… : nous filtrons les lignes du *DataFrame* résultat selon l'opérateur donnée.

5 Annexe : 10 exemples d'application des fonctions d'agrégation

Nous testons notre module DataLog sur la base *master.txt (présent dans notre archive de projet).* Nous vous invitons à d'ailleurs ce fichier par vous-même à travers la commande : *py main.py master.txt.*

- eleve(id, nom)
- prof(id, nom)
- ue(id, nom)
- inscription(id_eleve, id_ue)
- responsable(id_prof, id_ue)
- notes(id_eleve, id_ue, note)

Sur cette base, des requêtes de base nous donnent par exemple la liste des élèves inscrits par UE, ou la liste des UE donnés par les profs :

inscrits(NomUE, NomEleve):-ue(IdUE, NomUE), inscription(IdEleve,IdUE), eleve(IdEleve, NomEleve).
donne_cours(NomProf, NomUe):-ue(IdUE, NomUe), responsable(IdProf, IdUE), prof(IdProf, NomProf).

Nous pouvons également obtenir la liste des notes par UE, élève ou prof, dont le résultat pourra être utilisé avec des fonctions d'agrégation.

notes_par_ue(NomUe, Note):-note(_, IdUe, Note), ue(IdUe, NomUe).
notes_par_eleve(NomEleve, Note):-note(IdEleve, _, Note), eleve(IdEleve, NomEleve).
notes_par_prof(NomProf, Note):-note(_, IdUe, Note), responsable(IdProf, IdUe), prof(IdProf, NomProf).

Les résultats des requêtes ci-dessus se retrouvent directement en testant notre programme *master.txt*. On peut maintenant tester les fonctions d'agrégations.

5.1 Fonction COUNT

La fonction COUNT permet de compter le nombre de ligne d'un résultat. Voilà plusieurs exemples d'utilisation de la fonction COUNT sur notre base :

Nombre d'inscrits par UE :
nb_inscrits(NomUe, C):-inscrits(NomUe, NomEleve), COUNT(NomEleve, C).

Nombre d'UE suivies par élève :
nb_ue_suivies(NomEleve, C):-inscrits(NomUe, NomEleve), COUNT(NomUe, C).

Nombre de cours donnés par prof :
nb_cours_donné(NomProf, C):-donne_cours(NomProf, NomUe), COUNT(NomUe, C).

Nombre de copies à corriger par prof (nombre de notes données) :
nb_copie_prof(NomProf, C):-notes_par_prof(NomProf, Note), COUNT(Note, C).

```
nb_inscrits(NomUe, Count):
apprentissage_et_analyse_de_données 4
         base_de_données_déductives 3
                       base_de_l'ia 3
                        compilation 4
      conception_d'applications_web 4
                      cryptographie 1
gestion_de_projet_et_genie_logiciel 4
              informatique_graphique 3
                 logiciels_educatifs 2
                            reseaux 4
                   théorie_des_jeux 3

nb_ue_suivies(NomEleve, Count):
      julien  9
        loic 10
       loris 10
pierre_alain  6
```

```
nb_cours_donné(NomProf, Count):
      bonifati 1
       bouakaz 1
      elghazel 1
         gavin 2
guerin_lassous 1
  jean_daubias 1
       lefevre 1
        medini 1
           moy 2

nb_copie_prof(NomProf, Count):
      bonifati  3
       bouakaz  9
      elghazel  8
         gavin  5
guerin_lassous 12
  jean_daubias  6
       lefevre  6
        medini  8
           moy  9
```

5.2 Fonction AVG

La fonction AVG permet de faire la moyenne des lignes d'un résultat. Voilà plusieurs exemples d'utilisation de la fonction COUNT sur notre base :

Moyenne par élève et par UE :
eleve_moyenne_par_ue(NomEleve, NomUe, A):-eleve(IdEleve, NomEleve), note(IdEleve, IdUe, Note), ue(IdUe, NomUe), AVG(Note, A).

Moyenne par UE :
moyenne_ue(NomUe, A):-notes_par_ue(NomUe, Note), AVG(Note, A).

Moyenne générale par élève :
eleve_moyennes(NomEleve, A):-eleve_moyenne_par_ue(NomEleve, NomUe, Moyenne), AVG(Moyenne, A).
eleve_moyenne(NomEleve, A):-eleve_moyennes(NomEleve, Moyenne), AVG(Moyenne, A).

```
eleve_moyenne_par_ue(NomEleve, NomUe, Avg):
       loris gestion_de_projet_et_genie_logiciel 17.000000
       loris              informatique_graphique 13.666667
       loris       conception_d'applications_web 12.500000
       loris apprentissage_et_analyse_de_données 14.000000
       loris                           reseaux 10.666667
       loris                       base_de_l'ia 14.000000
       loris                        compilation 20.000000
       loris          base_de_données_déductives 15.000000
       loris                    théorie_des_jeux 14.000000
       loris                 logiciels_educatifs 19.333333
        loic gestion_de_projet_et_genie_logiciel 12.000000
        loic              informatique_graphique 12.333333
        loic       conception_d'applications_web 11.000000
        loic apprentissage_et_analyse_de_données 15.500000
        loic                           reseaux 16.333333
        loic                       base_de_l'ia 15.500000
        loic                        compilation 19.000000
        loic          base_de_données_déductives 16.000000
        loic                    théorie_des_jeux 17.000000
        loic                 logiciels_educatifs 20.000000
      julien gestion_de_projet_et_genie_logiciel 15.000000
      julien              informatique_graphique 12.666667
      julien       conception_d'applications_web 14.000000
      julien apprentissage_et_analyse_de_données 13.500000
      julien                           reseaux 10.333333
      julien                       base_de_l'ia 16.000000
      julien                        compilation 20.000000
      julien          base_de_données_déductives 12.000000
      julien                    théorie_des_jeux 15.000000
pierre_alain gestion_de_projet_et_genie_logiciel  9.000000
pierre_alain       conception_d'applications_web 11.000000
pierre_alain apprentissage_et_analyse_de_données 12.000000
pierre_alain                           reseaux 18.333333
pierre_alain                        compilation 17.000000
pierre_alain                      cryptographie 16.500000
```

```
eleve_moyenne(NomEleve, Avg):
      julien 14.277778
       loic 15.466667
      loris 15.016667
pierre_alain 13.972222

moyenne_ue(NomUe, Avg):
apprentissage_et_analyse_de_données 13.750000
         base_de_données_déductives 14.333333
                       base_de_l'ia 15.166667
                        compilation 19.000000
       conception_d'applications_web 12.125000
                      cryptographie 16.500000
gestion_de_projet_et_genie_logiciel 14.000000
              informatique_graphique 12.888889
                 logiciels_educatifs 19.666667
                           reseaux 13.916667
                     théorie_des_jeux 15.333333
```

5.3 Fonctions MIN/MAX

Les fonctions MIN/MAX permettent de récupérer le minimum/maximum des lignes d'un résultat. Voilà plusieurs exemples d'utilisation des fonctions MIN/MAX sur notre base :

Note min/max obtenue par UE :
note_max_ue(NomUe, M):-notes_par_ue(NomUe, Note), MAX(Note, M).
note_min_ue(NomUe, M):-notes_par_ue(NomUe, Note), MIN(Note, M).*

Note min/max obtenue par élève :
note_max_eleve(NomEleve, M):-notes_par_eleve(NomEleve, Note), MAX(Note, M).
note_min_eleve(NomEleve, M):-notes_par_eleve(NomEleve, Note), MIN(Note, M).

Note min/max donné par chaque prof :
note_max_prof(NomProf, M):-notes_par_prof(NomProf, Note), MAX(Note, M).
note_min_prof(NomProf, M):-notes_par_prof(NomProf, Note), MIN(Note, M).

```
note_max_ue(NomUe, Max):
apprentissage_et_analyse_de_données 16
         base_de_données_déductives 16
                    base_de_l'ia 18
                     compilation 20
      conception_d'applications_web 17
                    cryptographie 20
gestion_de_projet_et_genie_logiciel 18
             informatique_graphique 18
                logiciels_educatifs 20
                           reseaux 20
                théorie_des_jeux 17

note_min_ue(NomUe, Min):
apprentissage_et_analyse_de_données  8
         base_de_données_déductives 12
                    base_de_l'ia 11
                     compilation 17
      conception_d'applications_web  5
                    cryptographie 13
gestion_de_projet_et_genie_logiciel  9
             informatique_graphique  7
                logiciels_educatifs 18
                           reseaux  5
                théorie_des_jeux 14
```

```
note_max_eleve(NomEleve, Max):
      julien 20
       loic 20
      loris 20
pierre_alain 20

note_min_eleve(NomEleve, Min):
      julien 5
       loic 8
      loris 6
pierre_alain 5
```

```
note_max_prof(NomProf, Max):
      bonifati 16
       bouakaz 18
      elghazel 16
         gavin 20
guerin_lassous 20
  jean_daubias 20
       lefevre 18
        medini 17
           moy 20

note_min_prof(NomProf, Min):
      bonifati 12
       bouakaz  7
      elghazel  8
         gavin 13
guerin_lassous  5
  jean_daubias 18
       lefevre 11
        medini  5
           moy  9
```

5.4 Fonction SUM

La fonction SUM permet de faire la somme des lignes d'un résultat. Voilà plusieurs exemples d'utilisation de la fonction SUM sur notre base :

Nombre de points obtenus au total par élève :
nb_point_donné_prof(NomProf, S):-notes_par_prof(NomProf, Note), SUM(Note, S).

Nombre de points donnés au total par prof :
nb_point_obtenu_eleve(NomEleve, S):-notes_par_eleve(NomEleve, Note), SUM(Note, S).

```
nb_point_donné_prof(NomProf, Sum):
      bonifati  43
       bouakaz 116
      elghazel 110
         gavin  79
guerin_lassous 167
  jean_daubias 118
       lefevre  91
        medini  97
           moy 146

nb_point_obtenu_eleve(NomEleve, Sum):
      julien 218
       loic 294
      loris 295
pierre_alain 160
```

5.5 Opérateurs numériques

Nos moteurs Datalog prend en compte les opérateurs ‘==’, ‘!=’, ‘>’, ‘<’, ‘>=’, ‘<=’.

Toutes les notes supérieures à 18 :
noteSupA18(NomE, Note):- eleve(Id, NomE) note(Id, _, Note), Note>18.

```
noteSupA18(NomE, Note):
       loris 20
       loris 20
       loris 20
        loic 19
        loic 20
        loic 20
        loic 20
      julien 20
pierre_alain 20
pierre_alain 20
```

L’ID de l’élève portant le nom Loris :
iDLoris(Id):- eleve(Id, NomE), NomE==loris.

```
iDLoris(Id):
1
```