TP Project

Développement d'une extension d'un moteur **Datalog** pour les **fonctions d'aggrégations**.

Base de données déductives - M1 - UCBL - 2022/2023

Angela Bonifati

Etudiants :
Roméo PHANG
Romane LEDRU

I - Partie 1 : Preprocessing

1.1 - Choix du profil datalog

1 - Les fichiers de datalog sont dans le dossier /tests et peuvent être de format .dl
2 - Deux parties distinctes du fichier doivent être présentes:
- Les EDB, un par ligne.
- Les IDB, un par ligne, et pré-stratifiés (si un IDB en utilise un autre il se placera après), avec un `:-` caractérisant leur définition (entre la head et le body).

3 - Les prédicats commencent tous par une majuscule et sont séparés d'une virgule seulement. Les variables sont elles en majuscule pour ne pas se confondre à une valeur statique. (cf. annexe 1)
Structure imposée: pas de *not*.

Pour les fonctions d'aggrégations on ajoutera dans les IDB des prédicats avec (x, y) dont le x est l'indice utilisé dans la fonction et le y le nom de la fonction contenant les valeurs retournées. Il peut y avoir 1 ou plusieurs x (paramètres sur lesquels on va grouper). (cf. annexe 2)

Fonctions d'aggrégations disponibles par exemple:
- Count(x, Count) / Count(w, x, Count)
- Min(x, Min) / Min(w, x, Min)
- Max(x, Max) / Max(w, f, x, Max)
- Sum(x, Sum)/ Sum(w, x, Sum)
- Avg(x, Avg) / Avg(w, x, Avg)

Fonctions de comparaisons disponibles :
- E1 < E2
- E1 =< E2
- E1 > E2
- E1 >= E2
- E1 =:= E2 (valeur de E1 = valeur de E2, une autre façon de faire une comparaison d'égalité stricte mais en l'explicitant)
- E1 =\= E2 (valeur de E1 != valeur de E2)

On laissera le soin à l'utilisateur de créer un datalog correct avec les contraintes ci-dessus.

Différents fichiers dans le dossier /test ont été préparés afin de tester le programme dans différentes configurations, libre au correcteur de les utiliser pour visualiser rapidement l'exécution.

1.2 - Parsing

parse_input_file.py permet de prendre en entrée le fichier et itérer sur chacune de ses lignes afin d'en extraire les EDB d'un côté, et les IDB split avec la Head et le Body de l'autre.

Choix de l'implémentation en objet avec des class Edb.py, Idb.py, contenant des Predicate.py. Les paramètres sont en String ou Number. Les prédicats sont catégorisés s'ils sont des prédicats avec des comparaisons ou des prédicats d'aggrégations. (cf. annexe 3)

II - Partie 2 : Evaluation (cf. annexe 4)

Création du moteur d'évaluation pour notre datalog.

Pour ce faire, on va créer un fichier evaluation_progam.py qui prendra en entrée les données retournées de notre parser, et qui évaluera les IDB et retournera les réponses.

Le Dataframe

Point technique n°1 : Pour gérer les données, on parsera de nouveau les EDB dans un Dataframe.

Avantage : On n'aura pas à gérer à la main les jointures dans les IDBs.
Inconvénient : On rajoute une librairie sur le projet (pandas).

La récursivité

Point technique n°2 : On sait que dans un datalog on a parfois des IDB récursifs tels que : Ancestor(X,Y) :- Parent.., Ancestor(..). Pour gérer le cas de la récursivité, on évalue d'abord une première fois les IDBs, et on les ajoute à notre dataframe. Tant que la sortie n'appartient pas au dataframe (aka n'existe pas encore), on l'ajoute. Dès que la sortie existe déjà, on arrête la récursivité et on retourne le dataframe (cf. algo vu dans le cours).

L'évaluation d'une IDB

Point technique n°3 : On procède en 4 étapes :

1 - Récupération de tous les prédicats atomiques concernés par le body de l'IDB.
2 - On filtre d'abord en amont les valeurs statiques renseignées dans le body et on ne garde que les lignes concernées.
3 - On merge toutes les colonnes avec cross, c'est un merge force qui rend toutes nos combinaisons possibles sans même s'occuper de la jointure.
4 - Puis on s'occupe de la jointure: ça consiste à faire matcher les valeurs des colonnes, appelées tokens (cf. annexe 5), Z avec Z par exemple (merge_dataframes()). Un token est un nom de colonne unique que l'on souhaite filtrer, puis on retire les redondances (filterTable()) en loopant sur les tokens pour que les tokens identiques matchent en ayant la même valeur (cf. annexe 6).

Une fois le filtre récursivement exécuté sur tout le dataframe récupéré, on renvoie les valeurs qui nous intéressent (ceux qui sont présents dans la head de l'IDB) et on l'enregistre dans un fichier output.txt.

Les fonctions de comparaisons

Point technique n°4 : Pour les fonctions de comparaisons, on ira créer un Prédicat particulier, que l'on nommera ComparisonPredicate, qui héritera de ce qu'est un prédicat de base.
Il prendra 3 paramètres: une valeur, un opérateur de comparaison, et une valeur, dans cet ordre.
L'opérateur de comparaison fonctionnera tel qu'ils sont décrit dans la documentation datalog (cf. annexe 2), ils fonctionneront sur les entiers, et seul le =:= et le =\= fonctionnera sur les string.

Au sein du fichier Predicate.py, on spécifie directement de quel opérateur de comparaison il s'agit afin de le traiter directement à l'étape 4 de l'évaluation.

Les fonctions d'aggrégations

Point technique n°5 : Pour les fonctions d'aggrégations, on ira créer un Prédicat particulier, que l'on nommera AggregationPredicate, qui héritera de ce qu'est un prédicat de base.

Problématique : Comment, dans le cas où on a une fonction de comparaison sur une aggrégation, on ordonne l'évaluation ?
Solution : on a décidé d'ordonner dans l'ordre qui suit :
1 : les atomes de base,
2 : les comparaisons (colonne/colonne, colonne/valeur, valeur/colonne, valeur/valeur),
3 : les aggrégations,
4 : si la comparaison travaille sur une aggrégation, on le fait à la toute fin.
(cf.annexe 4)

```
%% Datalog test suite
%% This file contains a set of basic test cases for the Datalog interpreter.

% EDB (facts)

Actor(344759,"Douglas","Fowley").
Actor(355713,"William","Holden").
Actor(341002,"All","Movie").
Casts(344759,29851).
Casts(355713,29000).
Casts(341002,29000).
Casts(341002,7909).
Movie(7909,"A Night in Armour",1910).
Movie(29000,"Arizona",1940).
Movie(29445,"Ave Maria",1940).

% IDB

% 1. Find all the movies made in 1940.
MovieIn1940(Y):- Movie(_,Y,1940).

% 2. Find all Actors from the movie "Arizona".
ActorsInArizona(F,I):- Actor(Z,F,I),Casts(Z,X),Movie(X,"Arizona",_).

% 3. Find all the actors name who played in a movie made in 1940.
ActorsIn1940(F,I):- Actor(Z,F,I),Casts(Z,X),Movie(X,_,1940).

% 4. Find all the actors name who played in a movie made in 1940 and in a movie made in 1910.
ActorsIn1940And1910(F,I):- Actor(Z,F,I),Casts(Z,X1),Movie(X1,_,1940),Casts(Z,X2),Movie(X2,_,1910).
```

Annexe 1 : exemple de fichier datalog présent dans /tests.

4.7.10 Datalog Aggregates

This section lists both aggregate functions (to be used in expressions) and aggregate predicates (including grouping).

4.7.10.1 Aggregate Functions

Aggregate functions can only occur in the context of a `group_by` aggregate predicate (see next section) and apply to the result set for its input relation.

- `count(Variable)`

Return the number of tuples in the group so that the value for *Variable* is not null.

- `count`

Return the number of tuples in the group, disregarding tuples may contain null values.

- `sum(Variable)`

Return the sum of values for *Variable* in the group, ignoring nulls.

- `times(Variable)`

Return the product of values for *Variable* in the group, ignoring nulls.

- `avg(Variable)`

Return the average of values for *Variable* in the group, ignoring nulls.

- `min(Variable)`

Return the minimum value for *Variable* in the group, ignoring nulls.

- `max(Variable)`

Return the maximum value for *Variable* in the group, ignoring nulls.

4.7.10.3 Aggregate Predicates

- `count(Query,Variable,Result)`

Count in *Result* the number of tuples in the group for the query *Query* so that *Variable* is a variable of *Query* (an attribute of the result relation set) and this attribute is not null. It returns 0 if no tuples are found in the result set.

- `count(Query,Result)`

Count in *Result* the total number of tuples in the group for the query *Query*, disregarding whether they contain nulls or not. It returns 0 if no tuples are found in the result set.

- `sum(Query,Variable,Result)`

Sum in *Result* the numbers in the group for the query *Query* and the attribute *Variable*, which should occur in *Query*. Nulls are simply ignored.

- `times(Query,Variable,Result)`

Compute in *Result* the product of all the numbers in the group for the query *Query* and the attribute *Variable*, which should occur in *Query*. Nulls are simply ignored.

- `avg(Query,Variable,Result)`

Compute in *Result* the average of the numbers in the group for the query *Query* and the attribute *Variable*, which should occur in *Query*. Nulls are simply ignored.

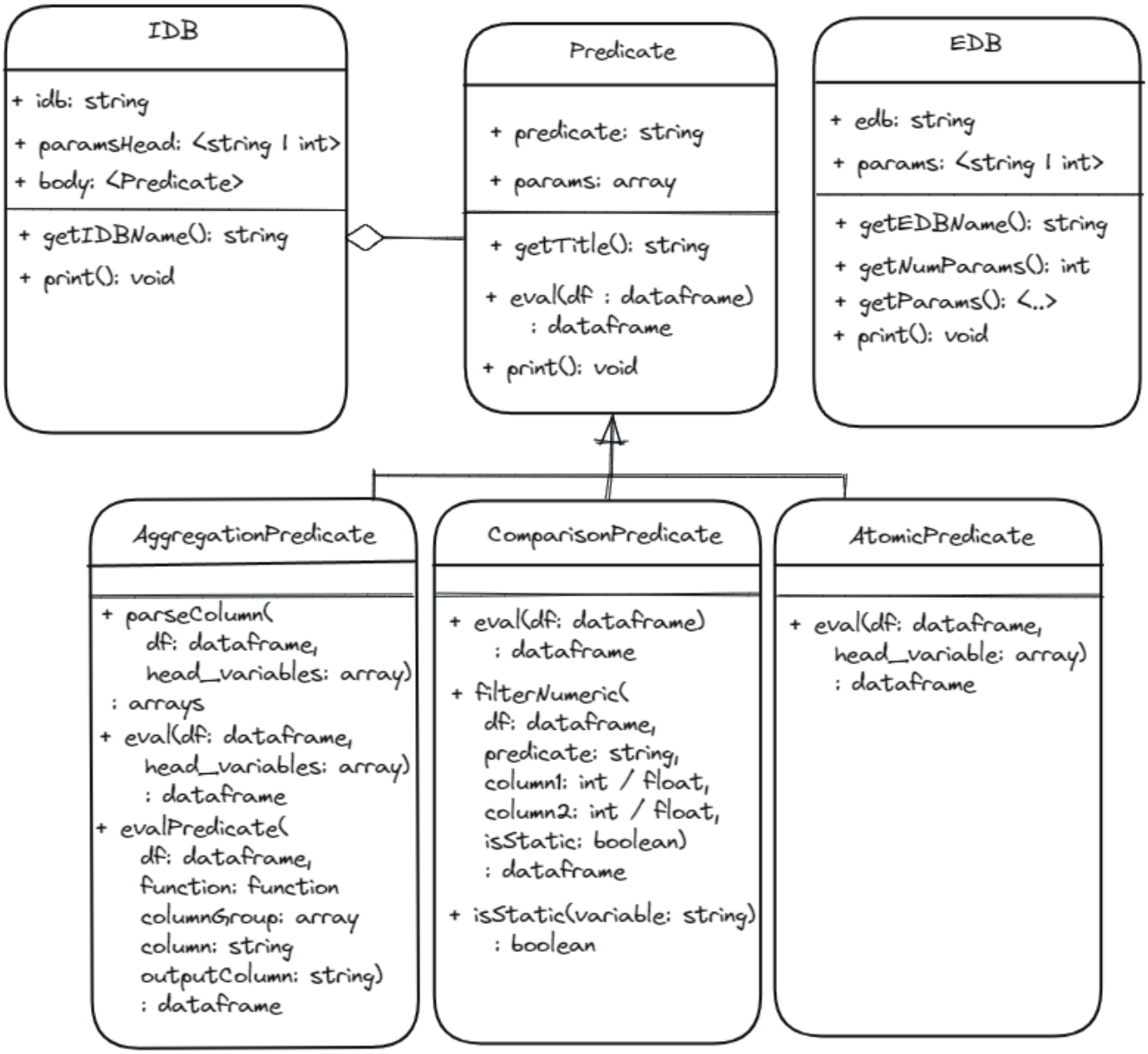
- `min(Query,Variable,Result)`

Compute in *Result* the minimum of the numbers in the group for the query *Query* and the attribute *Variable*, which should occur in *Query*. Nulls are simply ignored. If there are no such numbers, it returns `null`.

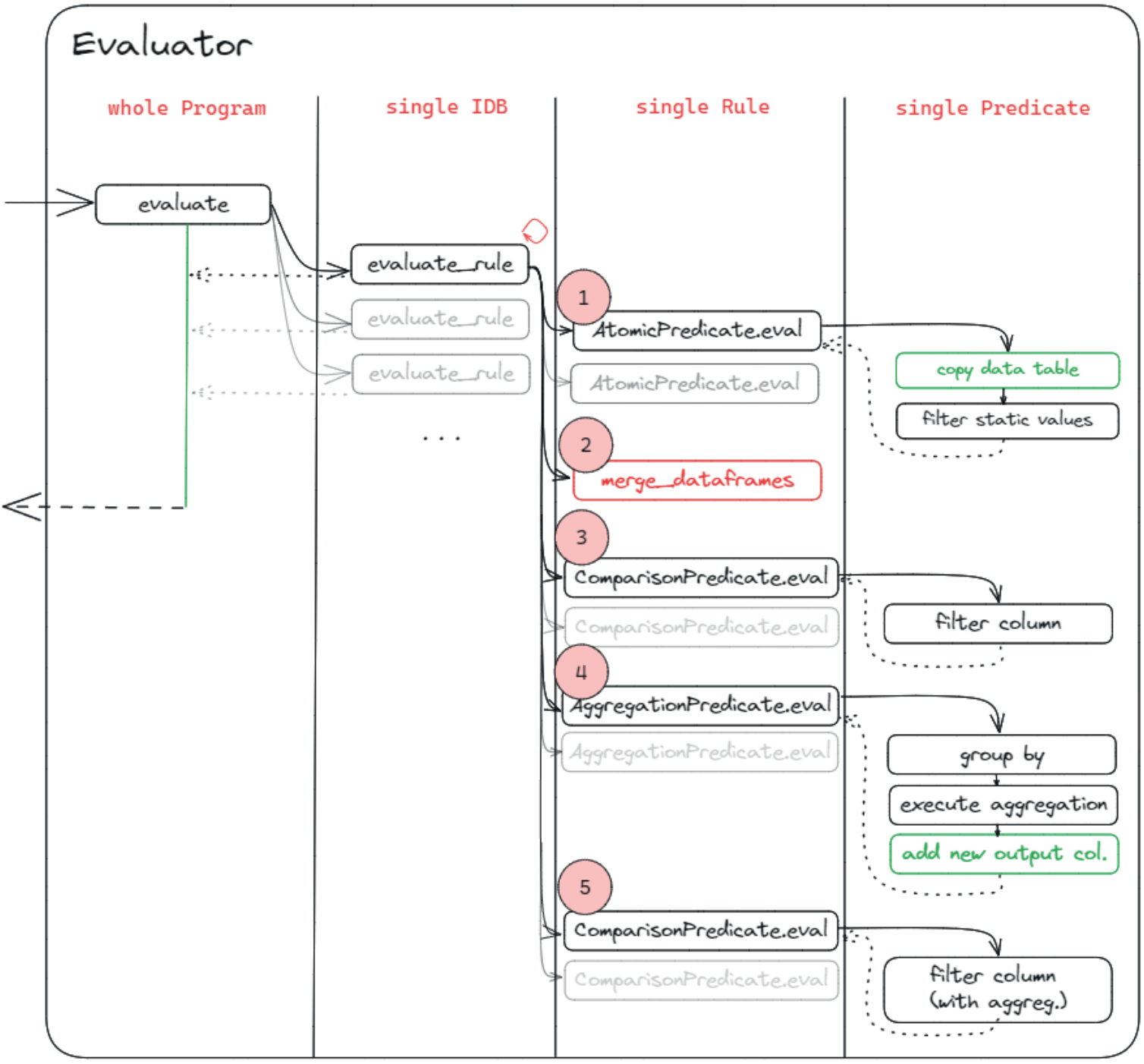
- `max(Query,Variable,Result)`

Compute in *Result* the maximum of the numbers in the group for the query *Query* and the attribute *Variable*, which should occur in *Query*. Nulls are simply ignored. If there are no such numbers, it returns `null`.

Annexe 2 : Documentation d'inspiration pour le fonctionnement des aggregations.



Annexe 3: structure objet post parser.



Annexe 4: schéma d'évalutation et appels de fonctions.

	#Z#_x	F	I	#Z#_y	#X#_x	#X#_y	#1940#
0	344759	Douglas	Fowley	344759	29851	29000	1940
1	344759	Douglas	Fowley	344759	29851	29445	1940
2	344759	Douglas	Fowley	355713	29000	29000	1940
3	344759	Douglas	Fowley	355713	29000	29445	1940
4	344759	Douglas	Fowley	341002	29000	29000	1940
5	344759	Douglas	Fowley	341002	29000	29445	1940
6	344759	Douglas	Fowley	341002	7909	29000	1940
7	344759	Douglas	Fowley	341002	7909	29445	1940
8	355713	William	Holden	344759	29851	29000	1940
9	355713	William	Holden	344759	29851	29445	1940
10	355713	William	Holden	355713	29000	29000	1940
11	355713	William	Holden	355713	29000	29445	1940
12	355713	William	Holden	341002	29000	29000	1940
13	355713	William	Holden	341002	29000	29445	1940
14	355713	William	Holden	341002	7909	29000	1940
15	355713	William	Holden	341002	7909	29445	1940
16	341002	All	Movie	344759	29851	29000	1940
17	341002	All	Movie	344759	29851	29445	1940
18	341002	All	Movie	355713	29000	29000	1940
19	341002	All	Movie	355713	29000	29445	1940
20	341002	All	Movie	341002	29000	29000	1940
21	341002	All	Movie	341002	29000	29445	1940
22	341002	All	Movie	341002	7909	29000	1940
23	341002	All	Movie	341002	7909	29445	1940

Annexe 5: Brut force du dataframe, avec comme nom de colone les tokens à merger.

	#Z#	F	I	#X#	#1940#
10	355713	William	Holden	29000	1940
20	341002	All	Movie	29000	1940

Annexe 6: resultat post-merge et donc résultat de la requête.