

# 1. Introducere

Obiectivul temei este implementarea în Prolog a unei biblioteci simple pentru reprezentarea, citirea și interogarea de tabele.

## 1.1. Reprezentarea tabelelor

Tabelele vor fi reprezentate în Prolog folosind predicatul `table_name(Name, Entries)`, unde **Name** este legat la un atom reprezentând numele tabelului, iar **Entries** este legat la o listă. În aceasta din urmă, primul element este lista numelor coloanelor din tabel, fiind urmată de lista valorilor din fiecare coloană. Exemplu:

```
table_name(students, [[id,name,class,aa,pp],
                      [1,"Matei Popovici","322",9,6],
                      [19,"Mihai","322",10,8],
                      [2020,"Dan","323",7,10]]).
```

Tabelele se vor **regăsi direct drept liste Prolog în fișierul ‘tables.pl’** furnizat de noi. Pentru a le încărca, puteți utiliza directiva `:- use_module(Files).` în fișierele ce conțin implementările voastre.

## 1.2. Afisarea tabelelor

O linie dintr-un tabel poate fi afișată folosind predicatul:

```
format(Str, Row)
```

built-in în Prolog, unde **Str** este legată la un **String** ce conține parametrii de formatare, iar **Row** este linia pe care dorim să o afișăm.

Pentru a construi variabila **Str** este nevoie să determinăm lungimea maximă a unei intrări de pe fiecare coloană. În exemplul anterior, pentru coloana **name**, aceasta este data de lungimea sirului **"Matei Popovici"**.

Predicatul `make_format_str`, a cărui implementare poate fi găsită în Anexa:

```
make_format_str(MaxRowLen, Str)
```

construiește sirul de formatare **Str**, plecând de la o listă de lungimi maxime, câte una pentru fiecare coloană din tabel. Pentru tabelul de mai sus, **MaxRowLen = [4, 14, 3, 2, 2]**.

Pentru a afisa un tabel, trebuie sa:

- Scrieti un predicat care determina **MaxRowLen** pentru tabelul in cauza
- Calculati sirul de formatare folosind **make\_format\_str**
- Folositi predicatul **format** impreuna cu sirul de formatare (mereu acelasi), pentru a afisa, succesiv, fiecare rand din tabel.

Implementati predicatul **print\_table\_op(Tbl)** care afiseaza un tabel folosind informatiile de mai sus.

### 1.3. Interogarea tabelelor.

#### (I) Table joins

Implementati predicatul:

**join\_op(Op, NewCols, T1, T2, R)**

Unde:

- **Op** este un predicat **p([X1, ... Xn], [Y1, ... Ym], [R1, ..., Rk])**.
- **NewCols** este o lista de **nume** de coloane avand lungime k.
- **T1** si **T2** sunt tabele avand n respectiv m coloane.
- **R** este tabelul rezultat din aplicarea predicatului **Op** pentru *join*-ul lui **T1** cu **T2**.

*Exemplu.* Daca **p** este predicatul:

**p([X],[Y],[Sum,Dif]) :- Sum is X + Y, Dif is X - Y.**

Atunci, **join\_op(p, [sums,difs], T1, T2, R)** va construi tabelul avand doua coloane (cu numele **sums** si **difs**), plecand de la **T1** avand o coloana (valorile X) si **T2** avand o coloana (valorile Y). Se considera ca tabelele au mereu acelasi numar de intrari (randuri).

#### (II) Select

Implementati predicatul:

**select\_op(T, Cols, R)**

care selecteaza din tabelul **T**, doar acele coloane din lista **Cols**. Rezultatul este legat la **R**. Este garantat ca toate coloanele din **Cols** apar in aceeasi ordine ca in tabel.

*Exemplu.* Scopul: **select\_op(T, [name,aa], R)**, pentru tabelul T din exemplul de mai sus, va lega variabila R la tabelul:

```
[[name,aa],
 ["Matei Popovici",9],
 ["Mihai",10],
 ["Dan",7]]
```

### (III) Filter

Implementati predicatul:

**filter\_op(T,Vars,Pred,R)**

Unde:

- **T** este tabelul cu **n** coloane pentru care realizam filtrarea
- **Vars** este o lista de **n** variabile **neinstantiate**, pozitia unui variabile in lista desemneaza coloana aferenta acesteia.
- **Pred** este un predicat de filtrare, exprimat in raport cu variabilele din **Vars**
- **R** va fi legata la tabelul filtrat, dupa ce predicatul satisface.

*Exemplu:* Apelul **filter\_op(T,[\_,\_,\_AA,PP],(AA + PP > 10),R)**, unde **T** este tabelul din exemplul anterior va elimina toate intrarile pentru care scopul **(AA + PP > 10)** nu este satisfacut.

*Observatie:* In absenta variabilelor din **Vars**, care trebuie sa ramana neinstantiate dupa verificarea unei linii, nu putem referi valorile concrete din fiecare linie, pentru a testa **Pred** pe acestea.

## 2. Query language

Pentru a **codifica** un query complex, ce combina operatiile descrise mai sus, folosim urmatoarele predicate descrise recursiv mai jos:

```
<table_query> ::= table(<table_name>)
                tfilter(Schema,Goal,<table_query>)
                select(COLUMNS, <table_query>)
                join(<table_query>,<table_query>)
                tprint(<table_query>)
```

Unde:

- **table(Str)** se evalueaza la tabelul avand numele **Str**
- **tfilter(S,G,Q)** evalueaza query-ul **Q** apoi filtreaza rezultatul intors.
- **select(COLUMNS, Q)** selecteaza anumite coloane din tabelul rezultat din query-ul **Q**
- **join(Pred, Cols, Q1, Q2)** realizeaza operatia de join pe cele doua tabele rezultate din **Q1** si **Q2**
- **tprint(Q)** afiseaza tabelul rezultat din query-ul **Q**.

Definiti predicatul:

**eval(Query, Tbl)**

Care implementeaza evaluarea unui query.

*Exemplu 1:*

```
eval(tprint(select([name, pp],
                  tfilter([_,_,_,_,PP], (PP >= 8),
                  table(students))))) , R)
```

**R** va fi legata la tabelul ce contine doar coloanele **name** si **pp** din rezultatul filtrarii tabelului **students** cu conditia **PP >= 8**.

Exemplul 2:

**avgg([X],[Y],[R]) :- R is (X + Y)/2.**

```
eval(tprint(join(avgg, [average],
                 select([aa], table(students)),
                 select([pp], table(students))))) , R)
```

**R** va fi legata la un tabel cu o singura coloana (cu numele **average**) ce contine media notelor la **aa** si **pp**.

### 3. Complex queries:

- **complex\_query1(Table, R)**

Selecteaza numele studentilor ce au media notelor la **AA** si **PP** mai mare ca 6, media tuturor notelor mai are ca 5 si care au prefixul „-escu” in numele de familie.

- **complex\_query2(Genre, MinRating, MaxRating, Table, R)**

Selecteaza filmele ce se incadreaza intr-un anumit gen si care au rating-ul aflat intre doua limite specificate.

Evaluarea query-urilor complexe se va realiza folosind tot predicatul **eval**.

## 4. Punctaj:

Tema va avea in total 100 de puncte, impartite astfel pe implementarea corecta a evaluarii query-urilor:

- [10p] `tprint`
- [10p] `select`
- [20p] `join`
- [20p] `tfilter`
- [20p] `complex_query1`
- [20p] `complex_query2`

Cele 100 de puncte se vor scala cu 1.5 pentru a se obtine punctajul final pe tema.

## 5. Rulare checker

Formatul comenzii pentru rularea checker-ului este urmatorul:

```
python3 check_test.py [--hwfile <file_name>.pl]
                        [--testname <test_name>]
                        [--reffile <reffile_path>]
                        [--testsetname <test_set_name>]
```

By default, se ruleaza toate testele, iar numele fisierului cu implementarile se considera implicit ca fiind **'main.pl'**. Fisierul de referinta trebuie specificat in cazul in care se doreste rularea unui singur test. De asemenea, se poate opta si pentru rularea unui anume set de teste (seturile sunt impartite per query).

Arhiva contine si un fisier **'check\_predicates.pl'**, in care se afla predicatele necesare anumitor teste. Pe acestea le puteti fie copia in fisierul cu implementarea voastra, fie puteti include modulul folosind directiva `:- use_module(check_predicates) ..`

## 6. Anexa:

```
plus5(X,Y):- Y is X + 5.
make_format_str(MaxRowLen,Str) :-
    maplist(plus5,MaxRowLen,Rp), aux_format(Rp,Str).
aux_format([H],R) :- string_concat("~t~w~t~",H,R1),
                    string_concat(R1,"+~n",R),
```

```
! .  
aux_format([H|T],R) :- string_concat("~t~w~t~",H,R1),  
                        string_concat(R1,"+ ",R2),  
                        aux_format(T,Rp),  
                        string_concat(R2,Rp,R) .
```