

# Building queries with Haskell

## 1. Introducere

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

### 1.1. Reprezentarea tabelelor

În exemplul următor avem un tabel:

```
-----  
|user_id|age|sex|occupation|zone |  
-----  
| 1      |24 |M  |technician|85711|  
| 2      |53 |F  |other     |94043|  
| 3      |23 |M  |writer    |32067|  
| 4      |24 |M  |technician|43537|  
| 5      |33 |F  |other     |15213|  
-----
```

Pentru a reprezenta tabele în Haskell, folosim următoarele structuri de date. Un *table schema* reprezintă o listă de **nume de coloane**, fiecare având un tip de valori asociat. În cele ce urmează vom considera ca toate tipurile de coloane sunt întotdeauna **String**:

```
type Column = String  
type TableSchema = [Column]
```

Pentru tabelul din exemplu de mai sus, *table schema* este reprezentat de lista:

```
["user_id", "age", "sex", "occupation", "zone"].
```

Un **Field** (un câmp) reprezintă o intrare dintr-o coloana a unui tabel:

```
type Field = String
```

Un *entry* reprezintă o linie din tabel (câte o valoare pentru fiecare câmp), și va fi reprezentată ca o listă de field-uri:

```
type Entry = [Field]
```

Spre exemplu, al doilea *entry* din tabelul de mai sus este:

```
["2", "53", "F", "other", "94043"].
```

Reprezentarea unui tabel conține un *table schema*, și o listă de *entry-uri*:

```
data Table = Table TableSchema [Entry]
```

## 1.2. Citirea tabelelor

Citirea unui tabel se face dintr-un String. Pentru a citi un tabel, trebuie să implementați funcția **read\_table** având următoarea semnătură:

```
read_table :: ColSeparator -> LnSeparator -> String -> Table
```

Unde:

```
type ColSeparator = Char
```

```
type LnSeparator = Char
```

Funcția primește un separator pentru coloane, un separator pentru linii și un string și întoarce tabelul codificat în acesta.

## 1.3. Afișarea tabelelor

Pentru a afișa tabele, vom folosi un format special. El este ilustrat în exemplul de mai jos.

```
-----  
|user_id|age|sex|occupation|zone |  
-----  
|1      |24 |M  |technician|85711|  
|2      |53 |F  |other     |94043|  
|3      |23 |M  |writer    |32067|  
|4      |24 |M  |technician|43537|  
|5      |33 |F  |other     |15213|  
-----
```

Tabela va fi bordată cu caracterele ‘-’ (pentru linii) și ‘|’ (pentru coloane).

Pentru a adauga corespunzător spații albe unui *Entry*, trebuie calculată lungimea maximă a unei valori, pentru fiecare coloană. (inclusiv numele coloanei din *TableSchema*).

În tabela de mai sus, *Field-urile* care ocupă cel mai mult spațiu sunt: *user\_id*, *age*, *sex*, *occupation* (sau *technician*) și *85711* (sau orice alt cod din *entry-uri*) -- acestea nu vor avea padding. Pentru celelalte *Field-uri* se va introduce un padding egal cu diferența dintre dimensiunea maximă găsită pe coloana respectivă și valoarea *Field-ului* de afișat.

Spre exemplu, în coloana “occupation”, dimensiunea maximă a unei valori este 10 (valoarea este *technician*). Așadar valoarea “other” va avea un padding de 5 spații albe.

#### 1.4. Interogarea tabelelor. Query-uri.

Query-urile **se evaluează** la tabele existente, dar și la altele noi, care sunt construite prin: **selectarea** anumitor coloane dintr-un tabel sau **selectarea unui număr limitat** de Entry-uri; **filtrarea de** Entry-uri, **combinarea** Entry-uri din mai multe tabele, **cosine-similarity**, care va fi discutat în detaliu mai jos.

În implementarea voastră, un query este reprezentat de tipul **Query**, care are următorii constructori:

(I) **Atom Table** – reprezintă exact tabelul primit ca parametru. Spre exemplu, **evaluarea:**

```
eval $ Atom user_info
```

va produce rezultatul:

-----				
user_id	age	sex	occupation	zone
-----				
1	24	M	technician	85711
2	53	F	other	94043
3	23	M	writer	32067
4	24	M	technician	43537
5	33	F	other	15213
-----				

(II) **Select [Field] Query**. Constructorul primește o lista de coloane și un query **q**. Evaluarea acestuia **selectează** doar coloanele specificate în tabelul la care se evaluează **q**.

Spre exemplu:

```
eval $ Select ["movie_id"] $ Atom movie
```

produce:

-----	
movie_id	
-----	
1	
2	
3	
4	
5	
6	
-----	

**Atenție!** Se pot selecta și coloane într-o **ordine diferită decât cea din table**:

**Ex:** `eval $ Select ["title", "movie_id"] $ Atom movie`

(III) **SelectLimit [Field] Integer Query** - la fel ca **Select**, însă selectează un număr specificat de *Entry-uri*. Spre exemplu:

`eval $ SelectLimit ["movie_id"] 3 $ Atom movie`

produce:

```
-----  
|movie_id|  
-----  
|1       |  
|2       |  
|3       |  
-----
```

(IV) **Filter FilterCondition Query** - reprezintă filtrarea *Entry-urilor* pe baza unei valori de tip **FilterCondition**. Condițiile de filtrare sunt reprezentate de următorul tip:

```
data FilterCondition = Lt Field Integer    |  
                      Eq Field String     |  
                      In Field [String]   |  
                      Not FilterCondition
```

unde:

- **Lt** verifică dacă *Field* (convertit la Integer) este **exclusiv mai mic** decât o valoare Integer.
- **Eq** verifică egalitatea unui *Field* cu o valoare dată.
- **In** verifică apartenența unui câmp la o listă de valori.
- **Not** reprezintă negația unei condiții de filtrare.

Spre exemplu:

`eval $ Filter (Lt "rating" 3) $ Atom rating`

va produce:

```
-----  
|user_id|movie_id|rating|  
-----  
|22      |377      |1      |  
|244     |51       |2      |  
|166     |346     |1      |  
-----
```

**Query** :|| **Query** - reprezintă **reuniunea** *Entry-urilor* din evaluarea a două query-uri diferite. Este garantat că evaluarea celor 2 query-uri produce tabele cu același *TableSchema*. Exemplu:

```
eval $ (Filter (Lt "user_id" 5) $ Atom user_info) :||  
      (Filter (Not $ Lt "user_id" 940) $ Atom user_info)
```

Evaluarea acestui query va întoarce o tabela în care se vor găsi doar *Entry-uri* având utilizatori cu *user\_id* mai mic decât 5 sau mai mare decât 940.

user_id	age	sex	occupation	zone
1	24	M	technician	85711
2	53	F	other	94043
3	23	M	writer	32067
4	24	M	technician	43537
940	32	M	administrator	02215
941	20	M	student	97229
942	48	F	librarian	78209
943	22	M	student	77841

## 2. Cerințe

[1pct] Implementați funcția:

```
read_table :: ColSeparator -> LnSeparator -> String -> Table
```

care citește un tabel dintr-un string, explicată anterior. Pentru testare, folosiți string-urile: *user\_info\_str*, *movie\_str* din scheletul de cod, și generați tabelele aferente. Ordinea coloanelor nu trebuie schimbată.

[2pct] Înrolați tipul *Table* în clasa *Show* astfel încât afișarea unei tabele să arate în modul explicat mai sus.

[6pct] Definiți funcția:

```
eval :: Query -> Table
```

care primește ca parametru un *Query* și întoarce un tabel, conform explicațiilor anterioare.

- Atom Table - 1p
- Select Query - 1.5p
- SelectList Integer Query - 0.5p
- Filter FilterCondition Query - 2p
- Query :|| Query - 1p

[1pct] Folosind **eval**, implementați funcțiile care să realizeze următoare operații:

- Selectați câmpurile *user\_id* și *ocupație* din tabela *user\_info* cu proprietatea că toți utilizatorii selectați sunt din aceeași zona cu un user identificat prin *user\_id* primit ca parametru.

**same\_zone :: String -> Query**

- Selectați câmpurile *ocupație* și *zona* din tabela *user\_info* cu proprietatea că intrările din tabela descriu persoane de sub **x** ani sau de peste **y** de ani de sex masculin.

**male\_within\_age :: Integer -> Integer -> Query**

- Selectați câmpurile *user\_id* din tabela *user\_info* cu proprietatea că intrările din tabelă descriu persoane care sunt dintr-o lista de zone și cu ocupația dintr-o lista de ocupații primite ca parametri, toți cu o vârstă mai mică strict decât un threshold primit ca parametru.

**mixed :: [String] -> [String] -> Integer -> Query**

### 3. (2p) Bonus - query-ul “cosine”

Implementați metoda **eval** pentru query-ul prezentat mai jos.

**Cosine Query.** Pentru a explica acest query, ne vom referi la o tabelă de *rating-uri*, având următorul *TableSchema*: [**“user\_id”**,**“movie\_id”**,**“rating”**]. O intrare reprezintă o notă (rating) acordat de un utilizator unui film.

Să presupunem că, în apelul **Cosine q**, query-ul **q** va fi evaluat la tabela de mai jos. Acesta conține rating-uri acordate de 3 utilizatori, pentru 5 filme.

-----		
user_id	movie_id	rating
-----		
98	105	3
98	106	3
91	105	1
91	106	1
91	107	2
91	112	1
96	106	2
96	105	3
96	108	1
-----		

Dorim să comparăm similaritatea preferințelor dintre doi utilizatori, spre exemplu 91 și 98. Ratingurile acordate de aceștia pentru fiecare dintre cele 5 filme pot fi reprezentate ca doi vectori 5-dimensional, unde fiecare dimensiune reprezintă un film.

$$\begin{array}{c}
 \begin{array}{ccccc}
 & 105 & 106 & 107 & 108 & 112 \\
 X_{98} = & (3 & 3 & - & - & -) \\
 X_{91} = & (1 & 1 & 2 & - & 1)
 \end{array}
 \end{array}$$

În acest context, similaritatea preferințelor reprezintă *cât de apropiate sunt direcțiile* celor doi vectori de preferințe. Formal, dorim să calculăm *cosinusul unghiului* dintre cei doi vectori X și Y, a cărei formula analitică este:

$$\cos(\theta) = \frac{X \cdot Y}{\|X\| \cdot \|Y\|}$$

sau echivalent:

$$\cos(\theta) = \frac{x_1y_1 + x_2y_2 + \dots + x_ny_n}{\sqrt{(x_1^2 + x_2^2 + \dots + x_n^2)} \cdot \sqrt{(y_1^2 + y_2^2 + \dots + y_n^2)}}$$

unde n reprezintă numărul de dimensiuni. Aplicând această formulă pentru vectorii  $X_{98}$  și  $X_{91}$  (și luând valoarea 0 pentru fiecare film care nu a primit rating) vom obține:

$$\cos(\theta) = \frac{3 + 3}{\sqrt{9 + 9} \cdot \sqrt{1 + 1 + 4 + 1}} = 0.5345$$

Aplicând aceeași formula pentru a afla similaritatea preferințelor între  $X_{98}$  și  $X_{96}$  vom obține 0,9449, un scor mai mare, care arată că preferințele utilizatorilor 98 și 96 sunt mult mai apropiate

decat cele ale utilizatorilor 98 și 91. Acest lucru este intuitiv și din scorurile pe care 98 și 96 le-au acordat filmelor 105 și 106.

Pentru a evalua query-ul **Cosine q**, query-ul **q** trebuie să se evalueze la un tabel cu *TableSchema* egal cu `["user_id", "movie_id", "rating"]`.

În acest context, query-ul **Cosine q** se va evalua la o tabela având întotdeauna un *TableSchema* egal cu `["user_id1", "user_id2", "sim"]`, și care conține, pentru fiecare pereche de utilizatori, valoarea  $\cos(\theta)$  care măsoară similaritatea dintre cei doi utilizatori, calculată după formula de mai sus.

#### Atentie!

- Nu se realizează similaritatea dintre utilizatorii cu același id
- Intrările trebuie sortate **crescător lexicografic** după coloana `"user_id1"`
- Pentru a controla câte zecimale sunt afișate pentru valoarea de similaritate, folosiți funcția `showFFloat` din biblioteca Numeric
- Pentru a nu calcula de două ori aceeași similaritate pentru perechi de utilizatori `(id1, id2)` și `(id2, id1)`, în tabela rezultat va apărea doar similaritatea corespunzătoare perechii `(id1, id2)` cu `id1 < id2`.

## 4. Scheletul de cod

Vom lucra cu o variantă ușor modificată a bazei de date oferită de MovieLens[1]. Acest dataset conține informații despre utilizatori, filme și rating-uri. Pentru mai multe informații puteți consulta cele 3 fișiere unde sunt ținute aceste date: *UserInfo.hs*, *Movie.hs*, *Rating.hs*.

## 4. Referințe

[1] <https://grouplens.org/datasets/movielens/>

[2] [http://mlwiki.org/index.php/Cosine\\_Similarity](http://mlwiki.org/index.php/Cosine_Similarity)