Building queries with Haskell

1. Introducere

Obiectivul temei este implementarea în Haskell a unei biblioteci simple pentru reprezentarea, citirea şi 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), şi 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 conţine un *table schema*, şi 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 implementaţi funcţia **`read_table`** având următoarea semnătură:

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

Unde:

```
type ColSeparator = Char
type LnSeparator = Char
```

Funcţia primeşte un separator pentru coloane, un separator pentru linii şi un string şi întoarce tabelul codificat în acesta.

1.3. Afişarea tabelelor

Pentru a afişa 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) şi ‘|’ (pentru coloane).
Pentru a adauga corespunzător spaţii 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 spaţiu sunt: *user_id, age, sex, occupation (sau technician)* şi *85711 (sau orice alt cod din entry-uri)* -- acestea nu vor avea padding. Pentru celelalte *Field-uri* se va introduce un padding egal cu diferenţa dintre dimensiunea maximă găsită pe coloana respectivă şi valoarea *Field-ului* de afişat.

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

1.4. Interogarea tabelelor. Query-uri.

Query-urile **se evaluează** la tabele existente, dar şi 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 primeşte o lista de coloane şi 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       |
----------
```

Atenţie! Se pot selecta şi 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`** - reprezinta filtrarea *Entry-urilor* pe baza unei valori de tip **`FilterCondition`**. Condiţiile de filtrare sunt reprezentate de următorul tip:

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

unde:

- **`Lt`** verifica dacă *Field* (convertit la Integer) este **exclusiv mai mic** decât o valoare Integer.
- **`Eq`** verifică egalitatea unui *Field* cu o valoare dată.
- **`In`** verifică apartenenţa unui câmp la o listă de valori.
- **`Not`** reprezintă negaţia unei condiţii 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
- SelectLimit 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 campurile *ocupație* și *zona* din tabela *user_info* cu proprietatea că intrările din tabela descriu persoane cu o vârstă mai mare strict de **`x`** ani și mai mică strict de **`y`** ani, de sex masculin.

  ```
  male_within_age :: Integer -> Integer -> Query
  ```

- Selectați campurile *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 preferinţelor dintre doi utilizatori, spre exemplu 91 şi 98. Ratingurile acordate de aceştia pentru fiecare dintre cele 5 filme pot fi reprezentate ca doi vectori 5-dimensionali, unde fiecare dimensiune reprezintă un film.

	105	106	107	108	112
X_{98} =	3	3	-	-	-)
X_{91} =	1	1	2	-	1)

În acest context, similaritatea preferinţelor reprezintă *cât de apropiate sunt* ***direcţiile*** celor doi vectori de preferinţe. Formal, dorim sa calculăm *cosinusul unghiului* dintre cei doi vectori X şi Y, a cărui formula analitică este:

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

sau echivalent:

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

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

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

Aplicând aceeaşi formula pentru a afla similaritatea preferinţelor între X_{98} şi X_{96} vom obţine 0,9449, un scor mai mare, care arată că preferinţele utilizatorilor 98 şi 96 sunt mult mai apropiate decat cele ale utilizatorilor 98 şi 91. Acest lucru este intuitiv şi din scorurile pe care 98 şi 96 le-au acordat filmelor 105 si 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"]`**, şi care conţine, pentru fiecare pereche de utilizatori, valoarea $cos(\theta)$ care măsoară similaritatea dintre cei doi utilizatori, calculată după formula de mai sus.

Atentie!

- Nu se realizeaza similaritatea dintre utilizatorii cu acelaşi id
- Intrările trebuie sortate **crescator lexicografic** după coloana **`"user_id1"`**
- Pentru a controla cate zecimale sunt afişate pentru valoarea de similaritate, folosiţi funcţia **`showFFloat`** din biblioteca Numeric
- Pentru a nu calcula de două ori aceasi similaritate pentru perechi de utilizatori **`(id1,id2)`** şi **`(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 uşor modificată a bazei de date oferită de MovieLens[1]. Acest dataset conţine informaţii despre utilizatori, filme şi rating-uri. Pentru mai multe informaţii puteţi consulta cele 3 fişiere unde sunt ţinute aceste date: *UserInfo.hs, Movie.hs, Rating.hs*.

5. Checker

Pentru testare, vom folosi un Makefile care apelează un script de python. Se foloseşte astfel:

- Pentru a rula toate testele cu checkerul: make run_tests.
- Pentru a testa de mana, **make run_shell**, iar dacă compilează cu succes alegem setul de test (specificat în Main.hs) şi subtestul dorit pentru a vedea output-ul.

Trebuie să vă asigurați că aveți python3 instalat pe mașină și că folosiți un sistem de operare echivalent cu Ubuntu 18.04.

6. Referințe

[1] https://grouplens.org/datasets/movielens/
[2] http://mlwiki.org/index.php/Cosine_Similarity