COMP221[illegible]g Language Concepts Coursework

r Julian Rathke

[illegible]mester 2 2023/24

Introduction

In this coursework you are required to *design and implement* a domain specific programming language for querying simple Graph Data documents. There are many query languages for assorted data formats, the most famous perhaps being SQL for relational databases. You are welcome to research any existing query languages to inspire the design of your own language. If you want, you could stick closely to the syntax of an existing language. Of course, you are also more than welcome to go your own way and be as original and creative as you want: it is *YOUR* programming language, and your design decisions.

You are required to (1) invent an appropriate syntax and (2) write an interpreter (possibly using `Alex` and `Happy` for lexing and parsing). Your overall goal is :

> To design and implement a programming language for querying and manipulating simple Graph Data input files.

For the five example problems listed below, you will be required to produce a program, in your language, that solves each of the problems. These five programs, together with the Haskell sources for your interpreter, are the required deliverables for the first submission.

Please keep a record of all the sources you consult that influence your design, and include them in your programming language manual. The manual will be required for the second submission, along with programs for five additional unseen problems, which we will make public *after* the deadline for the first submission. You should anticipate that the additional problems will comprise of variations and combinations of the first five problems. You will find more procedural details at the end of this document.

The specification is deliberately loose. If we haven't specified something, it is a design decision for you to make: e.g. how to handle syntax errors, illegal inputs, whether to allow comments, support for syntax highlighting, compile time warnings, any type systems etc. A significant part (50%) of the mark will be awarded for these qualitative aspects, where we will also take into account the design of the syntax and reward particularly creative and clean solutions with additional marks. The remaining 50% of the mark will be awarded for correctly solving a number of problems using your programming language. The correctness of your solution will be checked with a number of automated tests. We will release an "automarker" script before the first deadline which will give you an indication as whether your programs will compile and run against our test harness.

The Input and Output Format

For each problem we will declare the name of one input file in a simple Graph Data format. The particular variant of Graph Data we will be using is a simplified version of the input format for Neo4j (see https://neo4j.com/docs/operations-manual/current/tools/neo4j-admin/neo4j-admin-import/#import-tool-header-format). A key design point of the Neo4j Graph Data model is that both the nodes of the graph and the edges between nodes (relationships) may contain both typed property data and labelling information.

The input file name will correspond to a file in the current working directory with an extension ".n4j". For example, if the problem input file is called "foo" and your interpreter is executed in directory "C:/Home  n the input will be at "C:/Home/Users/jr/STQL/foo.n4j". The input files wil format as follows:

The overriding Comma Separated Value (CSV). That is, each input file will contain a nun na separated fields. Input files contain a number nodes sets followed by c ips between these nodes. Each node set begins with a node header row t rty field names and types for that node set. Then follows all of the entries f er row. Similarly, a relationship begins with relationship header row that a rty field names and types for the relationship.

A node header row must be of the form (using regular expression syntax):

```
:ID (,<name> : <Type>)* (,:LABEL)?
```

[Corrected from `:ID, (<name> : <Type>)* (,:LABEL)`]

that is, we must declare an ID field. We have zero or more named typed field declarations representing property values of the node. Finally, we have an **optional** keyword `:LABEL` that allows us to tag nodes with a label list. We separate multiple labels with semi-colon.

Subsequent entries in the input file following a node header must match the format of the previous header row. For example,

```
:ID, colour:string, age : integer , active : boolean , :LABEL
vehicle1 , "Yellow", 8, true , Car
vehicle2 , "Blue" , 7, true , Car
vehicle3 , "Red" , 11, false , Car
vehicle4 , null, 2, null, Car;Imported
```

The ID values must use alphanumeric characters only. The field names, that is the `<name>` entries, must be alpha-numeric only. Types can be one of `string`, `integer`, or `boolean` only. Fields must be monomorphic in that, if a field name is used multiple times across the document then it must be given the same type in each case. Labels must be alphabetic characters only. The literal values range over `string` values that are delimited by quotes and contain only alphabetic characters, `integer` values that match the regexp "[+-]?[0-9]+" and `boolean` values `true` and `false`. The value `null` is also a valid value of any type.

There may be multiple such node sets in the input file. Please note that Identifier values for the nodes must be unique across the input file. An input file that has two node rows beginning with the same ID value is considered an error, even if it is a duplicate row.

Following the node sets, we have a number of relationship sets. A relationship header row must be of the form:

```
:START_ID (, <name> : <Type>)* , :END_ID , :TYPE
```

[Corrected from `:START_ID, (<name> : <Type>)* , :END_ID , :TYPE`]

that is, we state that this is a relationship between a source node with a given start ID and a target node with a given end ID. There are zero or more property fields associated with the relationship and a relationship has exactly one TYPE. Like labels, the TYPE values describe the nature of the relationship and must be alphabetic characters only. Unlike labels, we only ever have a single TYPE. Subsequent entries in the input file following a relationship header must match the format of the previous header row. For example,

```
:START_ID, speed : integer , :END_ID, :TYPE
vehicle2 , 30, vehicle3, CrashedInto
vehicle2 , 20 , vehicle1, CrashedInto
vehicle4 , 15, vehicle2,  CrashedInto
```

describes three relationships (edges in the graph). Two of the relationships have a source node `vehicle2` with targets `vehicle3` and `vehicle1`. The third relationship has source node `vehicle4` and target `vehicl` nd e.g. `vehicle4 -CrashedInto->vehicle2` to describe the presence of an ultiple such relationship sets in the input file.

Spaces are allo s in each row and blank rows in the input file are permitted and may be sk ssume in the stated problems that all input files will be well-formed in this rmat. The order in which entries in a node set appear is unimportant. The ts appear is unimportant. The order in which the typed property data on s is unimportant and finally, the order in which labels appear is unimpor

For each stated problem, the output should also be in the same simple Neo4j format. I will compare your outputs against expected outputs by parsing the output as a graph and use a graph comparison function.

The output should always be printed to Standard Out.

Problems

For every problem below, you may assume that we will place a simple Neo4j file in the same directory where we execute your interpreter. The file will always be compatible with the simple Neo4j format. You may assume that we will not require you to perform any additional operations on the literal values other than those indicated by the problems given below. For each problem I will provide one example input file and the expected output for that input. These are listed in the Appendix but will also be available as a zip file from the module website. We will test your solutions on input files different to these but all inputs used will satisfy the input format above.

Problem 1 - Simple Node Query

Given an input file named "access.n4j", representing personnel and guests with access to a building, output a graph that contains all of the nodes in the input graph that have label "Visitor" along with all nodes whose value for field "age" is less than or equal to 25. The output should be in graph format as above given as a single node set only. The header row need only contain the ID field, the single "age" property and the label. You should use `null` values for nodes with no "age" value.

Problem 2 - Simple Relationship Query

Given an input file named "tasks.n4j", representing the tasks on a construction job, output a graph that contains the whole of the input graph along with some additional relationship edges as follows. For all nodes n_T that are the target of some relationship that has a field named "priority" with value greater than or equal to 8 and for all nodes labelled "Staff" n_S that are the source of some relationship that has a field value named "available" with value `true`, include the extra relationship n_S `-PossiblyAllocated->` n_T in the output graph.

Problem 3 - Parametric Queries

Given an input file named "table.n4j" representing a graph of sports teams that contains data for the season's results, find a list of teams that are on the same points as a team that drew with another team that the first team lost to. That is, find a set of nodes n such that n' `-Beat->` n for some source n' say. For each such n, find all nodes n'' such that either n'' `-DrewWith->` n' `-Beat->` n, or n' `-DrewWith->` n'' and n' `-Beat->` n for some n'. Return as output a graph consisting of nodes n whose field value named "points" is non-null and equal to the field value named "points" of n''. Your output graph can be returned as a single node set with header row just the node ID and the property field "points".

Problem 4 - Graph Filtering

Given a file named [illegible]nting a graph of persons, their friendships and employers, find all persons wit[illegible] with "A", "B", or "C", (as nodes) who have friends older than themselves th[illegible]. The employer nodes are labelled according to their type of business and al[illegible]Name" and "age" fields. There are two relationship sets with types "IsFrie[illegible]The "IsFriend" relationship is not necessarily symmetric.

You should ret[illegible]e input graph containg all person nodes that have first names beginning [illegible] and older friends that don't work in a cafe along with all of these friend[illegible] the friends that work in a cafe. You should return the "IsFriend" relationship set between the remaining nodes in the subgraph also.

Problem 5 - Field Updates

Given a filed named "loyalty.n4j" containing a graph of customers of a number of businesses operating a joint loyalty scheme, find all persons who are customers of any business who recommended another customer of that same business. There are person and business node sets and there are two relationship sets with labels "Recommended" and "CustomerOf". The "CustomerOf" relationship have a property field named "reward" and the business nodes have a property field named "bonus". You should output an modified graph as follows. Output an updated graph in which for all persons p such that p `-CustomerOf->` b for some b and p `-Recommended->` q `-CustomerOf->` b for some q, we have updated both the reward field of p `-CustomerOf->` b and q `-CustomerOf->` b by incrementing them both by the value of the "bonus" field of the business node b. You should also remove the entire relationship set for `Recommended`.

First submission - due Thursday April 25th 4pm

You will be required to submit a zip file containing:

- the sources for the interpreter for your language, written in Haskell
- five programs, written in YOUR language, that solve the five problems specified above. The programs should be in files named `pr1.gql`, `pr2.gql`, `pr3.gql`, `pr4.gql`, `pr5.gql`.

We will compile your interpreter using the command `ghc Gql.hs` so you will need to include a file in your zip file called `Gql.hs` that contains a main function along with any other Haskell source files required for compilation. Prior to submission, you are required to make sure that your interpreter compiles **on a Unix machine with a standard installation of GHC (Version 8.10.7) or earlier**: if your code does not compile then you may be awarded 0 marks. Before submission, we will release a simple "automarking" script that will allow you to check if your code compiles and whether each of your programs passes a basic test.

You can use Alex and Happy for lexing and parsing but make sure that you include the generated Haskell source files obtained by running the `alex` and `happy` commands as well as the Alex and Happy source files. Alternatively you can use any other Haskell compiler construction tools such as parser combinator libraries. You are welcome to use any other Haskell libraries, as long as this is clearly acknowledged and the external code is bundled with your submission, so that it can compile on a vanilla Haskell installation.

Interpreter spec. Your interpreter is to take a file name (the program in your language) as a single command line argument. The interpreter should produce output on standard output (`stdout`) and error messages on standard error (`stderr`). For each problem, we will test whether your code performs correctly by using a number of tests. We only care about correctness and performance will not be assessed (within reason - our marking scripts will timeout after a generous period of time). You can assume that for the tests we will use correctly formatted input. For example, when assessing your solution for Problem 1 we will run

`./Gql pr1.gql`

in a directory whe own versions of `graph.n4j`. We will then compare the contents of `stdou` outputs. Whitespace and formatting is unimportant as long as the outpu escribed simple Neo4j output format and **contains no other text**. We v and compare them against expected outputs as Graphs. For that reason, o utput file does not matter.

Second submiss May 2nd 4pm

Shortly after the first deadline we will release a further five problems. Although they will be different from Problems 1-5, you can assume that they will be closely related, and follow the same input/output conventions. You will be required to submit two separate files.

First, you will need to submit a zip file containing programs (`pr6.gql`, `pr7.gql`, `pr8.gql`, `pr9.gql`, `pr10.gql`) written in your language that solve the additional problems. We will run our tests on your solutions and award marks for solving the additional problems correctly.

Second, you will be required to submit a 3 page report on your language in **pdf format** that explains the main language features, its syntax, including any scoping and lexical rules as well as additional features such as syntax sugar for programmer convenience, type checking, informative error messages, etc. In addition, the report should explain the execution model for the interpreter, e.g. what the states of the runtime are comprised of, your key data structures used, and how they are transformed during execution. Languages that support strong static typing and type safety with a formal specification are preferred. This report, together with the five programs will be evaluated qualitatively and your marks will be awarded for the elegance and flexibility of your solution and the clarity of the report.

Please note: **there is only a short period between the first and second submission**. I strongly advise preparing the report well in advance throughout the development of the coursework.

As you know, the coursework is to be done in groups of three. Only one submission per group is required. I don't need to know who is in which group at the first submission but as as part of the **second** submission we will require a declaration of who is in your group and how marks are to be distributed amongst the members of your group. You will receive all feedback and your marks by Friday May 30th. Please ensure that it is the same group member that submits for both the first and second submissions.

Marks. This coursework counts for 40% of the total assessment for COMP2212. There are a total of 40 marks available. These are distributed between the two submissions as follows:

You will receive the test results of Submission One prior to the second deadline but no marks will be awarded until after Submission Two.

After Submission Two we will award up to 20 marks for the qualitative aspects of your solution, as described in your programming language report. We will also award up to 20 marks for your solutions to the ten problems. For each problem there will be 2 marks available for functional correctness only.

You have the option of resubmitting the interpreter after receiving your testing results from Submission One. This will however incur a 50% penalty on the marks for functional correctness of all ten problems. Therefore, if you decide to resubmit your interpreter in the second submission the maximum possible total coursework mark is capped at 30 marks (20 for the report plus 10 for functional correctness).

Any late submission to either component will be treated as a late submission overall and will be subject to the standard university penalty of 10% per working day late.

A Problem 1 Example Input:

```
:ID, firstname:                                  : string, role:string, age:integer, :LABEL
jj23, "John", "                                  , 43, Staff
uh12, "Umar", "                                  ", 55, Staff
gt2, "Guillaume                                  cher", 23, Staff
jv9, "Jennifer"                                  putyHead", 49, Staff
ab23, "Adam", "                                  22, Staff

:ID, firstname:                                  :string, age:integer ,:LABEL
nj10, "Nigel", "Jackson", 16, Student
rw5, "Rebecca", "Watson", 15, Student
pp8, "Peter", "Potter", 17, Student
jd6, "Jing","Ding",16, Student
cr2, "Connor","Flaherty",15, Student


:ID, firstname:string, familyname: string, :LABEL
v1, "Ray","Wise",Visitor
v2, "Barbara","King",Visitor
v3, "Mei","Wu",Visitor
v4, "Anika","Sharma",Visitor

:START_ID, :END_ID, :TYPE
v1, uh12, IsVisiting
v2, nj10, IsVisiting
v3, jd6,  IsVisiting
v4, uh12, IsVisiting
```


A.1 Expected Output for this Input:

```
:ID, age:integer, :LABEL
gt2,  23, Staff
ab23, 22, Staff
nj10, 16, Student
rw5,  15, Student
pp8,  17, Student
jd6,  16, Student
cr2,  15, Student
v1,   null, Visitor
v2,   null, Visitor
v3,   null, Visitor
v4,   null, Visitor
```

B Problem 2 Example Input:

```
:ID, site:strin
loc1, "Garden",
loc2, "FrontRoo
loc3, "Kitchen"
loc4, "MainBedr

:ID, descriptio                integer, :LABEL
task1, "Paving"
task2, "Fencing",  12,  Job
task3, "Wiring",   4,   Job
task4, "Plumbing", 12,  Job
task5, "Painting", 4,   Job

:ID, name:string, :LABEL
emp1, "Jane",   Staff
emp2, "Bill",   Staff
emp3, "Winona", Staff
emp4, "Rajesh", Staff
emp5, "Jakub",  Staff

:START_ID, priority:integer, :END_ID, :TYPE
loc1, 2,  task1,  ToComplete
loc1, 4,  task2,  ToComplete
loc2, 9,  task3,  ToComplete
loc2, 1,  task5,  ToComplete
loc3, 8,  task3,  ToComplete
loc3, 10, task4,  ToComplete
loc4, 9,  task3,  ToComplete
loc4, 1,  task5,  ToComplete

:START_ID, available:boolean, :END_ID, :TYPE
emp1, true, task1, CanDo
emp1, true, task2, CanDo
emp2, false, task1, CanDo
emp3, true, task3, CanDo
emp4, true, task2, CanDo
emp4, true, task4, CanDo
emp5, false, task4, CanDo
```


B.1 Expected Output for this Input:

```
:ID, site:string, :LABEL
loc1, "Garden",      Location
loc2, "FrontRoom",   Location
loc3, "Kitchen",     Location
loc4, "MainBedroom", Location

:ID, description:string, duration:integer, :LABEL
task1, "Paving",   8,   Job
task2, "Fencing",  12,  Job
task3, "Wiring",   4,   Job
task4, "Plumbing", 12,  Job
```

```
task5, "Painting", 4,   Job

:ID, name:strin
emp1, "Jane",
emp2, "Bill",
emp3, "Winona",
emp4, "Rajesh",
emp5, "Jakub",

:START_ID, prio                    ID, :TYPE
loc1, 2,  task1,  ToComplete
loc1, 4,  task2,  ToComplete
loc2, 9,  task3,  ToComplete
loc2, 1,  task5,  ToComplete
loc3, 8,  task3,  ToComplete
loc3, 10, task4,  ToComplete
loc4, 9,  task3,  ToComplete
loc4, 1,  task5,  ToComplete

:START_ID, available:boolean, :END_ID, :TYPE
emp1, true, task1, CanDo
emp1, true, task2, CanDo
emp2, false, task1, CanDo
emp3, true, task3, CanDo
emp4, true, task2, CanDo
emp4, true, task4, CanDo
emp5, false, task4, CanDo

:START_ID, :END_ID, :TYPE
emp1, task3, PossiblyAllocated
emp1, task4, PossiblyAllocated
emp3, task3, PossiblyAllocated
emp3, task4, PossiblyAllocated
emp4, task3, PossiblyAllocated
emp4, task4, PossiblyAllocated
```

C Problem 3 Example Input:

```
:ID, team:strin
t1,  "Wincheste
t2,  "Romsey",
t3,  "Eastleigh
t4,  "FairOak",
t5,  "Totton",
t6,  "Weston",
t7,  "Hamble",
t8,  "Fareham",    null
t9,  "Ringwood",   null
t10, "Hythe",      3
t11, "Shirley",    3
t12, "Southampton",3
t13, "Ashurst",    4
t14, "Lyndhurst",  6

:START_ID, gf:integer, ga:integer , week:integer, :END_ID, :TYPE
t1,  3, 2, 3, t10, Beat
t11, 1, 0, 3, t2,  Beat
t3,  1, 1, 3, t12, DrewWith
t13, 2, 1, 3, t4,  Beat
t14, 3, 2, 3, t5,  Beat
t6,  1, 0, 3, t8,  Beat
t7,  1, 0, 3, t9,  Beat

t1,  1, 0, 2, t9,  Beat
t10, 1, 0, 2, t2,  Beat
t3,  4, 0, 2, t11, Beat
t4,  3, 3, 2, t12, DrewWith
t5,  0, 0, 2, t13, DrewWith
t14, 1, 0, 2, t6,  Beat
t7,  1, 1, 2, t8,  DrewWith

t1, 4, 2, 1, t8,  Beat
t2, 1, 1, 1, t9,  DrewWith
t3, 1, 0, 1, t10, Beat
t4, 3, 1, 1, t11, Beat
t5, 2, 2, 1, t12, DrewWith
t6, 1, 0, 1, t13, Beat
t7, 2, 0, 1, t14, Beat
```


C.1 Expected Output for this Input:

```
:ID, points:integer
t10,  3
t11,  3
```

D Problem 4 Example Input:

```
:ID, firstname:string , age:integer
jj23, "John", "
uh12, "Umar", "
gt2, "Guillaume
jv9, "Jennifer"
ab23, "Adam", "
nj10, "Nigel",
rw5, "Rebecca",
pp8, "Peter", "Potter", 17
jd6, "Jing","Ding", 16
rw11, "Ray","Wise", 66
bk21, "Barbara","King",70
mw3, "Mei","Wu",47
as4, "Anika","Sharma",40
cr2, "Connor","Flaherty",15

:ID, company:string, :LABEL
com1, "BarStucks", Cafe
com2, "NaffeCero", Delicatessen;Cafe
com3, "CoffeeNumberTwo", Cafe;Pizzeria
com4, "TreatyEats", Delicatessen
com5, "CrustInUs", Pizzeria;Delicatessen
com6, "DoughReMe", Pizzeria
com7, "BaaBaaBlackSheep", Barber
com8, "TheAngryOnion", Pizzeria;Restaurant
com9, "TheLaughingOnion", Restaurant

:START_ID, :END_ID, :TYPE
jj23,com5,WorksFor
uh12,com2,WorksFor
gt2,com1,WorksFor
jv9,com9,WorksFor
ab23,com8,WorksFor
nj10,com3,WorksFor
rw5,com4,WorksFor
pp8,com6,WorksFor
rw11,com7,WorksFor
bk21,com4,WorksFor
mw3,com2,WorksFor
as4,com1,WorksFor
cr2,com6,WorksFor

:START_ID, :END_ID, :TYPE
jj23,rw5,  IsFriend
jj23,bk21, IsFriend
uh12,as4,  IsFriend
uh12,jv9,  IsFriend
gt2,pp8,   IsFriend
gt2,jd6,   IsFriend
gt2,cr2,   IsFriend
jv9,as4,   IsFriend
jv9,uh12,  IsFriend
```

```
jv9,rw11,   IsFriend
ab23,cr2,   IsFriend
ab23,gt2,   IsFr
nj10,jd6,   IsFr
rw5,jj23,   IsFr
rw5,mw3,    IsFr
bk21,rw11,  IsFr
as4,gt2,    IsFr
as4,jv9,    IsFr
as4,mw3,    IsFr
cr2,gt2,    IsFriend
cr2,pp8,    IsFriend
```

D.1 Expected Output for this Input:


```
:ID, firstname:string, familyname:string , age:integer
jv9, "Jennifer", "Villeneuve", 49
pp8, "Peter", "Potter", 17
as4, "Anika","Sharma",40
cr2, "Connor","Flaherty",15
```



```
:START_ID, :END_ID, :TYPE
jv9,as4,    IsFriend
as4,jv9,    IsFriend
cr2,pp8,    IsFriend
```

E Problem 5 Example Input:

```
:ID, firstname:string , age:integer
jj23, "John", "
uh12, "Umar", "
gt2, "Guillaume
jv9, "Jennifer"
ab23, "Adam", "
nj10, "Nigel",
rw5, "Rebecca",
pp8, "Peter", "Potter", 17
jd6, "Jing","Ding", 16
rw11, "Ray","Wise", 66
bk21, "Barbara","King",70
mw3, "Mei","Wu",47
as4, "Anika","Sharma",40
cr2, "Connor","Flaherty",15

:ID, business:string, bonus:integer, :LABEL
com1, "BarStucks", 10, Cafe
com2, "NaffeCero", 15, Cafe
com3, "CoffeeNumberTwo", 20, Cafe
com4, "TreatyEats", 10, Delicatessen
com5, "CrustInUs", 15, Pizzeria
com6, "DoughReMe", 25, Pizzeria
com7, "BaaBaaBlackSheep", 10, Barber
com8, "TheAngryOnion", 30, Restaurant
com9, "TheLaughingOnion", 25, Restaurant

:START_ID, reward:integer, :END_ID, :TYPE
jj23,72,com5,CustomerOf
jj23,12,com2,CustomerOf
jj23,5,com3,CustomerOf
uh12,24,com8,CustomerOf
uh12,24,com2,CustomerOf
uh12,24,com5,CustomerOf
gt2,12,com1,CustomerOf
jv9,22,com9,CustomerOf
jv9,0,com5,CustomerOf
jd6,11,com1,CustomerOf
ab23,23,com8,CustomerOf
nj10,26,com3,CustomerOf
rw5,2,com4,CustomerOf
rw5,22,com3,CustomerOf
pp8,86,com6,CustomerOf
rw11,43,com7,CustomerOf
bk21,62,com4,CustomerOf
mw3,3,com2,CustomerOf
as4,63,com1,CustomerOf
cr2,50,com6,CustomerOf

:START_ID, :END_ID, :TYPE
jj23,rw5,  Recommended
jj23,bk21, Recommended
```

```
uh12,as4,  Recommended
uh12,jv9,  Recommended
gt2,jd6,   Reco
jv9,as4,   Reco
ab23,cr2,  Reco
nj10,jd6,  Reco
rw5,mw3,   Reco
bk21,rw11, Reco
as4,mw3,   Reco
cr2,gt2,   Reco
cr2,pp8,   Recommended
```

E.1 Expected Output for this Input:


```
:ID, firstname:string, familyname:string , age:integer
jj23, "John", "Jones", 43
uh12, "Umar", "Habib", 55
gt2, "Guillaume", "Truffaut", 23
jv9, "Jennifer", "Villeneuve", 49
ab23, "Adam", "Baker", 22
nj10, "Nigel", "Jackson", 16
rw5, "Rebecca", "Watson", 15
pp8, "Peter", "Potter", 17
jd6, "Jing","Ding", 16
rw11, "Ray","Wise", 60
bk21, "Barbara","King",70
mw3, "Mei","Wu",47
as4, "Anika","Sharma",40
cr2, "Connor","Flaherty",15

:ID, business:string, bonus:integer, :LABEL
com1, "BarStucks", 10, Cafe
com2, "NaffeCero", 15, Cafe
com3, "CoffeeNumberTwo", 20, Cafe
com4, "TreatyEats", 10, Delicatessen
com5, "CrustInUs", 15, Pizzeria
com6, "DoughReMe", 25, Pizzeria
com7, "BaaBaaBlackSheep", 10, Barber
com8, "TheAngryOnion", 30, Restaurant
com9, "TheLaughingOnion", 25, Restaurant

:START_ID, reward:integer, :END_ID, :TYPE
jj23,72,com5,CustomerOf
jj23,12,com2,CustomerOf
jj23,25,com3,CustomerOf
uh12,24,com8,CustomerOf
uh12,24,com2,CustomerOf
uh12,39,com5,CustomerOf
gt2,22,com1,CustomerOf
jv9,22,com9,CustomerOf
jv9,15,com5,CustomerOf
jd6,21,com1,CustomerOf
```