MONASH University
Information Technology

ASSESSMENT COVER SHEET

Student ID number	30708974
Given Name	Malav Arunkumar
Family name	Parikh

Unit Name and Code:	**FIT5137 Advance Database Technology**		
Campus:	**Clayton**		
Assignment Title:	**Assignment 2: Neo4j (Individual)**		
Name of Lecturer:	**Agnes Haryanto**		
Name of Tutor:	**Arif Hidayat and Shuyi Sun**		
Tutorial Day and Time:	**Tuesday 2-4 PM**		
Phone Number:			
Email Address:	**mpar0034@student.monash.edu**		
Has any part of this assignment been previously submitted as part of another unit/course? ☐ Yes ☐ No			
Due Date:	**06-11-2020**	Date Submitted:	**06-11-2020**

All work must be submitted by the due date. If an extension of work is granted this must be specified with the signature of the lecturer/tutor.

Extension granted until (date)____________________ Signature of lecturer/tutor ___________________________

Please note that it is your responsibility to retain copies of your assessments.

Intentional plagiarism or collusion amounts to cheating under Part 7 of the Monash University (Council) Regulations

Plagiarism: Plagiarism means taking and using another person's ideas or manner of expressing them and passing them off as one's own. For example, by failing to give appropriate acknowledgement. The material used can be from any source (staff, students or the internet, published and unpublished works).

Collusion: Collusion means unauthorised collaboration with another person on assessable written, oral or practical work and includes paying another person to complete all or part of the work.

Where there are reasonable grounds for believing that intentional plagiarism or collusion has occurred, this will be reported to the Associate Dean (Education) or delegate, who may disallow the work concerned by prohibiting assessment or refer the matter to the Faculty Discipline Panel for a hearing.

Student Statement:

- I have read the university's Student Academic Integrity Policy and Procedures.
- I understand the consequences of engaging in plagiarism and collusion as described in Part 7 of the Monash University (Council) Regulations http://adm.monash.edu/legal/legislation/statutes
- have taken proper care to safeguard this work and made all reasonable efforts to ensure it could not be copied.
- No part of this assignment has been previously submitted as part of another unit/course.
- I acknowledge and agree that the assessor of this assignment may for the purposes of assessment, reproduce the assignment and:
  i. provide to another member of faculty and any external marker; and/or
  ii. submit it to a text matching software; and/or
  iii. submit it to a text matching software which may then retain a copy of the assignment on its database for the purpose of future plagiarism checking.
- I certify that I have not plagiarised the work of others or participated in unauthorised collaboration when preparing this assignment.

Signature .... Date...

* delete (iii) if not applicable

FIT5137 Assignment 2

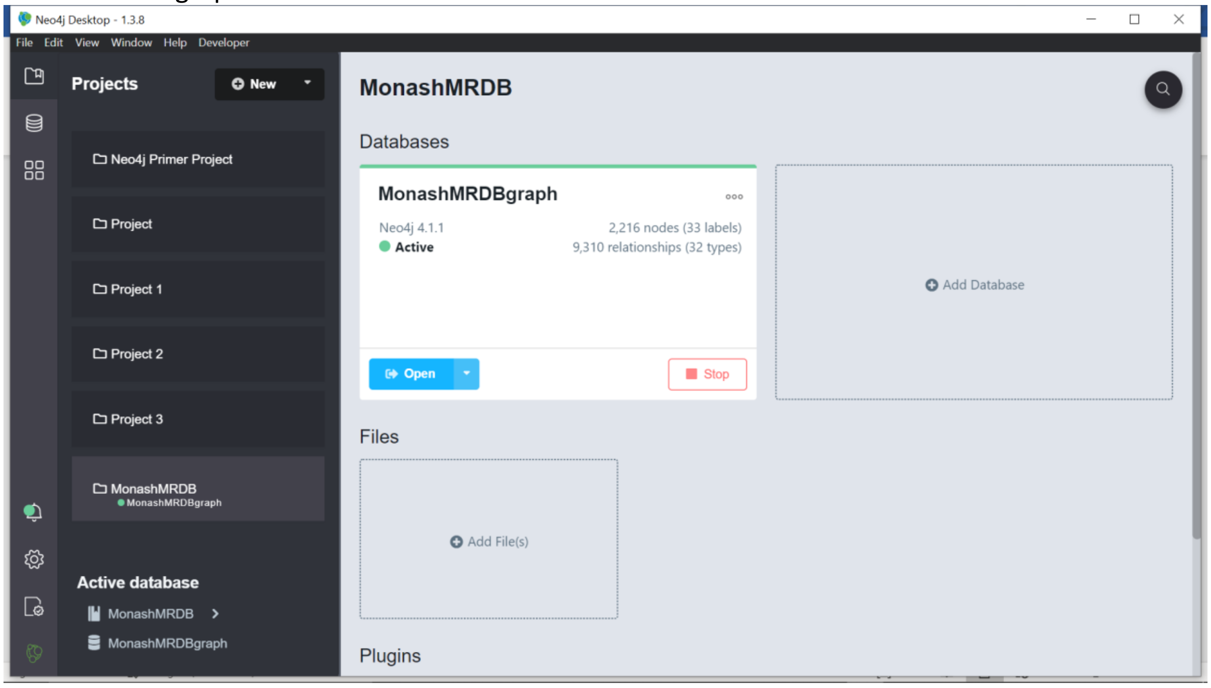
Tutor: Arif Hidayat and Shuyi Sun
Tutorial: Tuesday 2-4 PM

Malav Parikh 30708974

MONASH MRDB

C.1 DATABASE DESIGN

Create a new project called MonashMRDB
Create a new graph called MonashMRDBgraph
As you can see below the project named MonashMRDB is created and a database named MonashMRDBgraph has been added.



Identification of Nodes and Edges

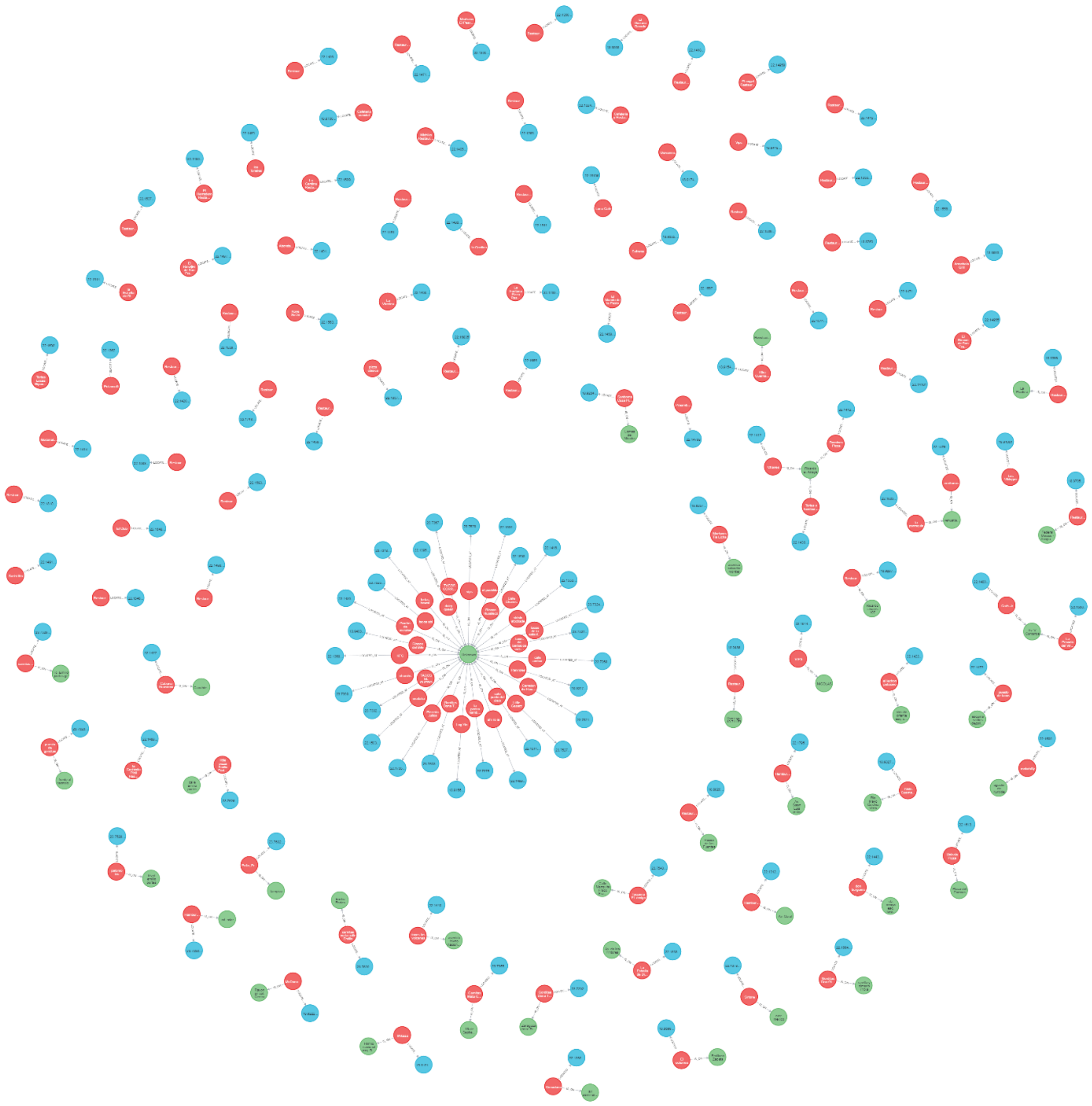
The data needs to be imported from five csv file namely:

- userProfile.csv
- placeProfiles.csv
- openingHours.csv
- user_ratings.csv
- place_ratings.csv

The user profile contains data about users, place profile contains data about places, opening hours about the time and day a place is open, user ratings contains data about users having rated or reviewed a place by giving it ratings and similarly place ratings contains data about places having received ratings.

The nodes that have been developed are:
Places: The node Places contains properties place Id and place name
Rest all the properties of a place have been divided up into **individual nodes** to reduce the redundancy of data and to remove the instance of a property being repeated in multiple nodes
Users: The node Users contains properties user Id and user personal traits such as weight, height and birth year. Rest all the properties have been divided up into **individual nodes** to reduce the redundancy of data and to remove the instance of a property being repeated in multiple nodes

Ratings: The node Ratings contains properties rating id, rating food, rating place, rating service and it has two foreign keys one is place Id which has been matched with the places node and the user Id which has been matched with the users node.
Opening Hours: The node opening hours contains hours and days data of a place operating on. It has place Id as foreign key which has been matched with the places node.



The above image shows a set of 300 nodes only due to the limitation of Neo4j Desktop.
The command used here is
MATCH (n) RETURN n

Different nodes are created for each of the following from the user profiles csv:
Latitude, longitude, marital status, interest, worker type, fav color, drink level, budget, smoker, dress preference, ambience, transport, religion, employment, cuisines and payment method

Different nodes are created for each of the following from the place profiles csv:

Latitude, longitude, state, street, city, country, alcohol, smoking, dress code, accessibility, price, franchise, area, other services, parking arrangements, payment modes and cuisines

Some nodes from both the files have been combined and added as one node having different labels like the price node has two labels: price and budget as places have prices and users have budget but both the properties have common values which makes it better to take them out as individual nodes instead of repeating it in all the nodes as properties.

As shown in the code below dress code or preference, prices or budget, payment modes or preferred payment methods and cuisines or fav cuisines have been made as one instead of repeating it for every node of user and place individually as a property.

```
MERGE (d:Dresscodes :DressPreferences {dress_code: line.`placeFeatures dress_code`})
MERGE (p)-[:HAS_DRESSCODE]->(d)

MERGE (pr:Prices :Budgets {`price or budget`: line.`placeFeatures price`})
MERGE (p)-[:HAS_PRICES]->(pr)

MERGE (pay:Payments {payment: paymentMode})
MERGE (p)-[:ACCEPTS]->(pay)

FOREACH (ignoreMe IN (CASE WHEN cuisine IS NULL THEN [] ELSE [1] END) |
   MERGE (cu:Cuisines { cuisines: cuisine})
   CREATE (p)-[:SERVES]->(cu)
)
```

The code below shows all the nodes and edges that have been created by importing data from the provided five csv:

```
// load data from places.csv

LOAD CSV WITH HEADERS FROM "file:///places.csv" AS line
WITH line,split(line.cuisines, ", ") AS cuisines,split(line.acceptedPaymentModes, ", ") AS
paymentModes WHERE line._id IS NOT NULL
UNWIND case coalesce(size(cuisines),0) when 0 then [null] else cuisines end
  as cuisine
UNWIND paymentModes as paymentMode
MERGE (p:Places {place_id: toInteger(line._id)})
ON CREATE SET p.place_name = line.placeName
MERGE (l:Locations { latitude: toFloat(line.`location latitude`), longitude: toFloat(line.`location
longitude`)})
MERGE (s:Streets {street : replace(line.`address street`,"?","Unknown")})
MERGE (c:Cities {city : replace(line.`address city`,"?","Unknown")})
MERGE (st:States {state : replace(line.`address state`,"?","Unknown")})
MERGE (co:Countries {country : line.`address country`})
MERGE (p)-[:LOCATED_AT]->(l)
MERGE (p)-[:IS_ON]->(s)
MERGE (p)-[:LOCATED_IN]->(c)
MERGE (p)-[:BELONGS_TO]->(st)
MERGE (p)-[:PRESENT_IN]->(co)
MERGE (sm:Smoking  {smoke: line.`placeFeatures smoking_area`})
MERGE (p)-[:SMOKING_IS]->(sm)
MERGE (al:Alcohol  {alcohol: line.`placeFeatures alcohol`})
MERGE (p)-[:ALCOHOL_IS]->(al)
MERGE (d:Dresscodes :DressPreferences {dress_code: line.`placeFeatures dress_code`})
MERGE (p)-[:HAS_DRESSCODE]->(d)
MERGE (a:Accessibility {accessibility: line.`placeFeatures accessibility`})
MERGE (p)-[:HAS]->(a)
MERGE (pr:Prices :Budgets {`price or budget`: line.`placeFeatures price`})
MERGE (p)-[:HAS_PRICES]->(pr)
MERGE (ar:Area {area: line.`placeFeatures area`})
MERGE (p)-[:HAS_AREA]->(ar)
MERGE (o:Services {other_services: line.`placeFeatures otherServices`})
MERGE (p)-[:HAS_SERVICES]->(o)
MERGE (pa:Parkings {parking: line.parkingArragements})
MERGE (p)-[:HAS_PARKING]->(pa)
MERGE (pay:Payments {payment: paymentMode})
MERGE (p)-[:ACCEPTS]->(pay)
FOREACH (ignoreMe IN (CASE WHEN cuisine IS NULL THEN [] ELSE [1] END) |
   MERGE (cu:Cuisines { cuisines: cuisine})
   CREATE (p)-[:SERVES]->(cu)
)
MERGE (f:Franchises {franchise: line.`placeFeatures franchise`})
MERGE (p)-[:HAS_FRANCHISE]->(f);

// replace f and t as false and true respectively and convert them to boolean

MATCH (f:Franchises {franchise: 't'})
SET f.franchise = toBoolean(replace('t','t','TRUE'))
RETURN f;

MATCH (f:Franchises {franchise: 'f'})
SET f.franchise = toBoolean(replace('f','f','FALSE'))
RETURN f;

// load data from userProfiles.csv

LOAD CSV WITH HEADERS FROM "file:///userProfile.csv" AS line
WITH line,split(line.favCuisines, ", ") AS cuisines,split(line.favPaymentMethod, ", ") AS
paymentModes WHERE line._id IS NOT NULL
UNWIND case coalesce(size(cuisines),0) when 0 then [null] else cuisines end
  as cuisine
UNWIND case coalesce(size(paymentModes),0) when 0 then [null] else paymentModes end
  as paymentMode
MERGE (u:Users {user_id: toInteger(line._id)})
ON CREATE SET u.birth_year = date(line.`personalTraits birthYear`),
u.weight = toInteger(line.`personalTraits weight`),
u.height = toFloat(line.`personalTraits height`)
MERGE (l:Locations {latitude: toFloat(line.`location latitude`), longitude: toFloat(line.`location
longitude`)})
```

```
MERGE (u)-[:LOCATED_AT]->(l)
FOREACH (ignoreMe IN (CASE WHEN line.`personalTraits maritalStatus` IS NULL THEN [] ELSE [1]
END) |
   MERGE (sta:Status {marital_status: line.`personalTraits maritalStatus`})
   CREATE (u)-[:IS]->(sta)
)
MERGE (i:Interests {interest: line.`personality interest`})
MERGE (u)-[:HAS_INTEREST_IN]->(i)
MERGE (t:Worker {typeofworker: line.`personality typeOfWorker`})
MERGE (u)-[:IS_A]->(t)
MERGE (col:Colors {color: line.`personality favColor`})
MERGE (u)-[:LIKES]->(col)
MERGE (dr:Drinkers {drinkLevel: line.`personality drinkLevel`})
MERGE (u)-[:HAS_DRINK_LEVEL]->(dr)
FOREACH (ignoreMe IN (CASE WHEN line.`preferences budget` IS NULL THEN [] ELSE [1] END) |
   MERGE (pr:Prices :Budget {`price or budget`: line.`preferences budget`})
   CREATE (u)-[:HAS_BUDGET]->(pr)
)
FOREACH (ignoreMe IN (CASE WHEN line.`preferences smoker` IS NULL THEN [] ELSE [1] END) |
   MERGE (sm:Smokes {smoker: line.`preferences smoker`})
   CREATE (u)-[:IS_SMOKER]->(sm)
)
FOREACH (ignoreMe IN (CASE WHEN line.`preferences dressPreference` IS NULL THEN [] ELSE [1]
END) |
   MERGE (d:Dresscodes :DressPreferences {dress_code: line.`preferences dressPreference`})
   CREATE (u)-[:PREFERS_DRESS]->(d)
)
FOREACH (ignoreMe IN (CASE WHEN line.`preferences ambience` IS NULL THEN [] ELSE [1] END) |
   MERGE (a:Ambiences {ambience: line.`preferences ambience`})
   CREATE (u)-[:PREFERS_AMBIENCE]->(a)
)
FOREACH (ignoreMe IN (CASE WHEN line.`preferences transport` IS NULL THEN [] ELSE [1] END) |
   MERGE (t:Transports {transport: line.`preferences transport`})
   CREATE (u)-[:PREFERS_MODE]->(t)
)
MERGE (r:Religions {religion: line.`otherDemographics religion`})
MERGE (u)-[:PREACHES]->(r)
FOREACH (ignoreMe IN (CASE WHEN line.`otherDemographics employment` IS NULL THEN [] ELSE [1]
END) |
   MERGE (e:Employments {employment: line.`otherDemographics employment`})
   CREATE (u)-[:PROFESSION]->(e)
)
FOREACH (ignoreMe IN (CASE WHEN cuisine IS NULL THEN [] ELSE [1] END) |
   MERGE (cu:Cuisines { cuisines: cuisine})
   CREATE (u)-[:LOVES]->(cu)
)
FOREACH (ignoreMe IN (CASE WHEN paymentModes IS NULL THEN [] ELSE [1] END) |
   MERGE (pay:Payments { payment: paymentMode})
   CREATE (u)-[:PAYS_BY]->(pay)
);
```

```
// load data from place_ratings.csv

LOAD CSV WITH HEADERS FROM "file:///place_ratings.csv" AS line
WITH line WHERE line.rating_id IS NOT NULL
MATCH (p:Places {place_id: toInteger(line.place_id)})
MATCH (u:Users {user_id: toInteger(line.user_id)})
MERGE (r:Ratings {rating_id: toInteger(line.rating_id)})
ON CREATE SET r.rating_place = toInteger(line.rating_place),
r.rating_food = toInteger(line.rating_food),
r.rating_service = toInteger(line.rating_service)
MERGE (p)-[:HAS_RATINGS]->(r)
MERGE (u)-[:GAVE]->(r);

// load data from user_ratings.csv
// it wont produce any changes as we keep only the properties as in place_ratings.csv to avoid
redundancy of data
// for each place properties and user properties

LOAD CSV WITH HEADERS FROM "file:///user_ratings.csv" AS line
WITH line WHERE line.rating_id IS NOT NULL
MATCH (p:Places {place_id: toInteger(line.place_id)})
MATCH (u:Users {user_id: toInteger(line.user_id)})
MERGE (r:Ratings {rating_id: toInteger(line.rating_id)})
ON CREATE SET r.rating_place = toInteger(line.rating_place),
r.rating_food = toInteger(line.rating_food),
r.rating_service = toInteger(line.rating_service)
MERGE (p)-[:HAS_RATINGS]->(r)
MERGE (u)-[:GAVE]->(r);

// load data from openingHours.csv

LOAD CSV WITH HEADERS FROM "file:///openingHours.csv" AS line
WITH line,split(line.hours, ";") AS hours, split(line.days, ";") AS days WHERE line.hours IS NOT NULL
and line.days IS NOT NULL
MATCH (p:Places {place_id: toInteger(line.placeID)})
MERGE (o:OpeningHours {hour: hours, day: days})
MERGE (p)-[:OPERATES]->(o);
```

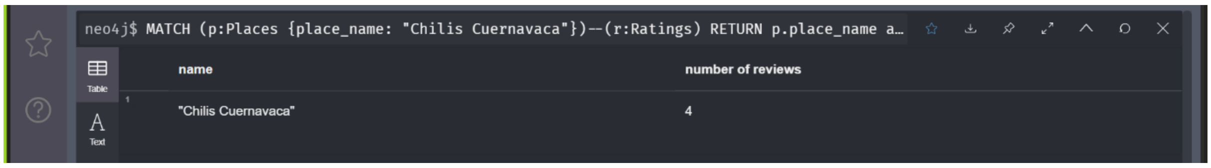
The data has been loaded by proper casting of data types and where they cant be casted have been converted later on using appropriate code like in the part of franchise where f and t have been converted to Boolean values after replacing them with false and true respectively.

C.2 QUERIES

1. How many reviews does "Chilis Cuernavaca" have?
Code:

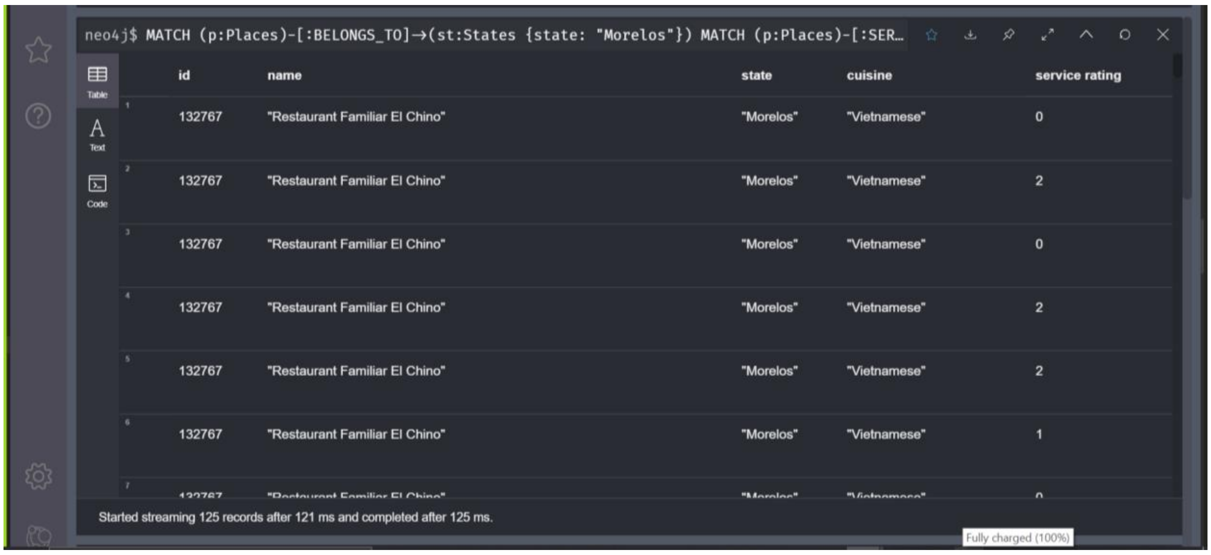
```
MATCH (p:Places {place_name: "Chilis Cuernavaca"})--(r:Ratings)
RETURN p.place_name as name, COUNT(*) as `number of reviews`;
```

Screenshot:



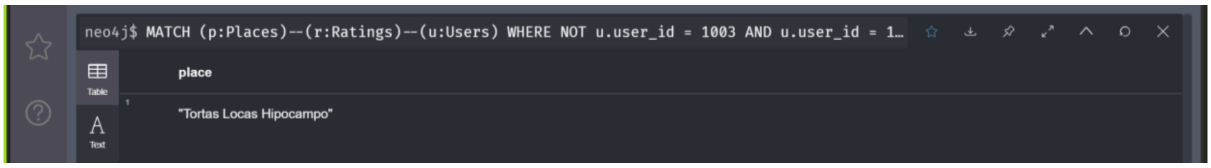
```
neo4j$ MATCH (p:Places {place_name: "Chilis Cuernavaca"})--(r:Ratings) RETURN p.place_name a…

     name                                   number of reviews
1    "Chilis Cuernavaca"                    4
```

2. Show all place, cuisines, and service ratings for restaurants in "Morelos" state.
Code:
MATCH (p:Places)-[:BELONGS_TO]->(st:States {state: "Morelos"})
MATCH (p:Places)-[:SERVES]->(c:Cuisines)
MATCH (p:Places)-[:HAS_RATINGS]->(r:Ratings)
RETURN p.place_id as id, p.place_name as name, st.state as state, c.cuisines as cuisine,
r.rating_service as `service rating`;
Screenshot:



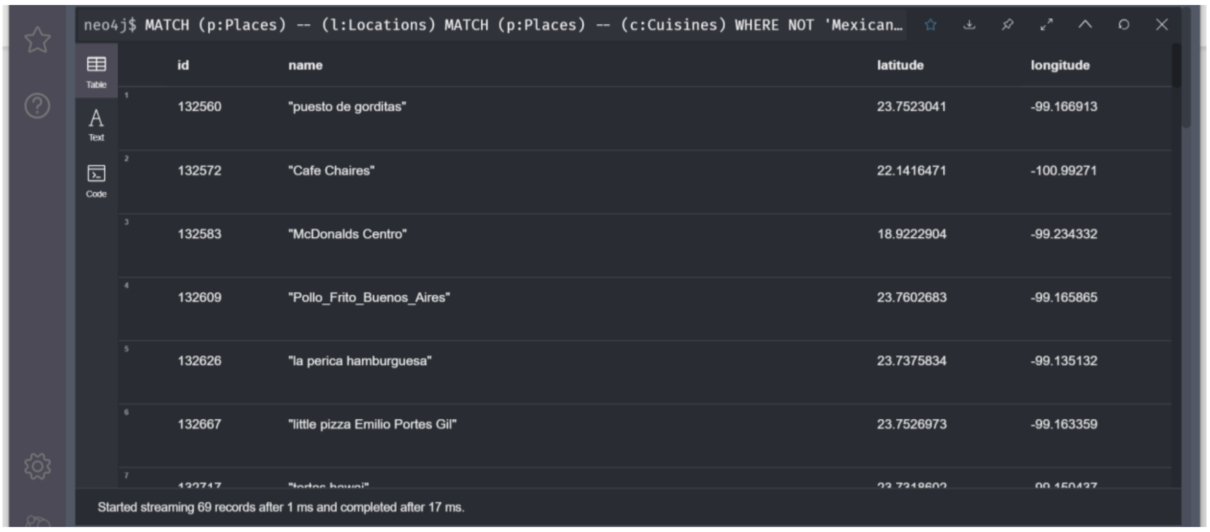
```
neo4j$ MATCH (p:Places)-[:BELONGS_TO]→(st:States {state: "Morelos"}) MATCH (p:Places)-[:SER…

     id       name                              state       cuisine        service rating
1    132767   "Restaurant Familiar El Chino"    "Morelos"   "Vietnamese"   0
2    132767   "Restaurant Familiar El Chino"    "Morelos"   "Vietnamese"   2
3    132767   "Restaurant Familiar El Chino"    "Morelos"   "Vietnamese"   0
4    132767   "Restaurant Familiar El Chino"    "Morelos"   "Vietnamese"   2
5    132767   "Restaurant Familiar El Chino"    "Morelos"   "Vietnamese"   2
6    132767   "Restaurant Familiar El Chino"    "Morelos"   "Vietnamese"   1
7    132767   "Restaurant Familiar El Chino"    "Morelos"   "Vietnamese"   0

Started streaming 125 records after 121 ms and completed after 125 ms.
```

3. Can you recommend places that user 1003 has never been but user 1033 have been and gave ratings above 1?
Code:
MATCH (p:Places)--(r:Ratings)--(u:Users)
WHERE NOT u.user_id = 1003 AND u.user_id = 1033 AND
r.rating_food > 1 AND r.rating_place > 1 AND r.rating_service > 1
RETURN p.place_name as place;
Screenshot:



```
neo4j$ MATCH (p:Places)--(r:Ratings)--(u:Users) WHERE NOT u.user_id = 1003 AND u.user_id = 1…

     place
1    "Tortas Locas Hipocampo"
```

4. List all restaurant names and locations that do not provide Mexican cuisines.
Code:
MATCH (p:Places) -- (l:Locations)
MATCH (p:Places) -- (c:Cuisines)
WHERE NOT 'Mexican' in c.cuisines
RETURN distinct p.place_id as id, p.place_name as name,
l.latitude as latitude, l.longitude as longitude;
Screenshot:

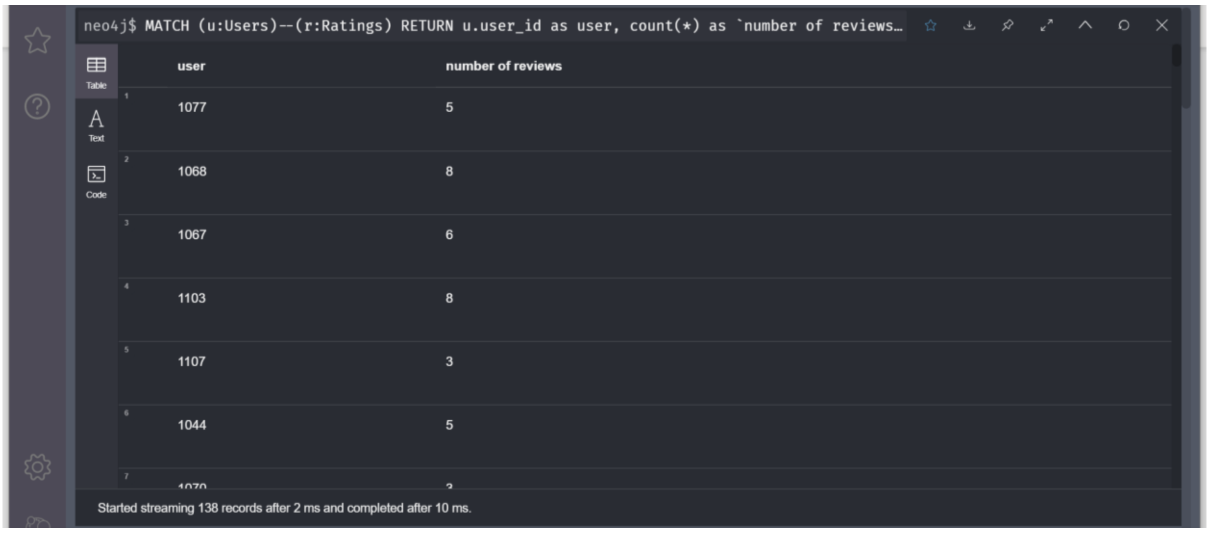


```
neo4j$ MATCH (p:Places) -- (l:Locations) MATCH (p:Places) -- (c:Cuisines) WHERE NOT 'Mexican…
```

	id	name	latitude	longitude
1	132560	"puesto de gorditas"	23.7523041	-99.166913
2	132572	"Cafe Chaires"	22.1416471	-100.99271
3	132583	"McDonalds Centro"	18.9222904	-99.234332
4	132609	"Pollo_Frito_Buenos_Aires"	23.7602683	-99.165865
5	132626	"la perica hamburguesa"	23.7375834	-99.135132
6	132667	"little pizza Emilio Portes Gil"	23.7526973	-99.163359

Started streaming 69 records after 1 ms and completed after 17 ms.

5. Count how many times each user provides ratings.
Code:
MATCH (u:Users)--(r:Ratings)
RETURN u.user_id as user, count(*) as `number of reviews`;
Screenshot:



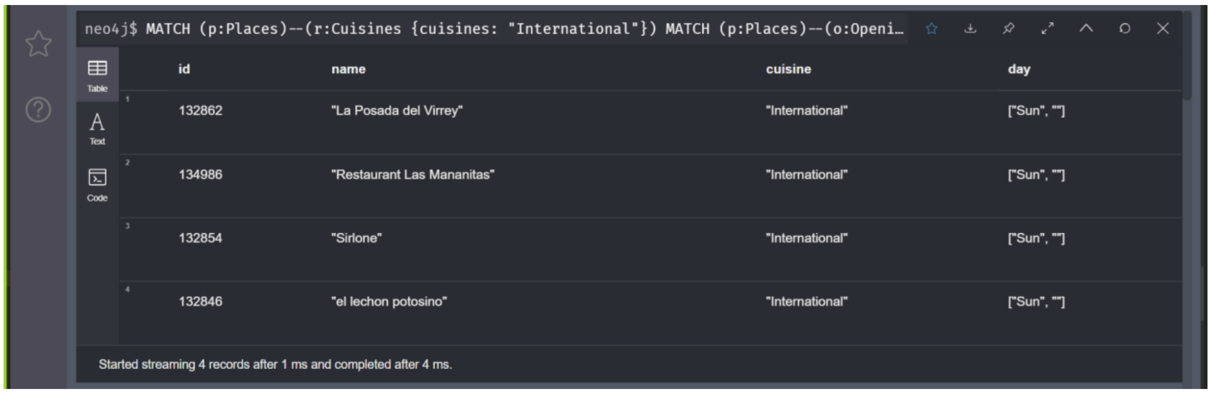
```
neo4j$ MATCH (u:Users)--(r:Ratings) RETURN u.user_id as user, count(*) as `number of reviews…
```

	user	number of reviews
1	1077	5
2	1068	8
3	1067	6
4	1103	8
5	1107	3
6	1044	5

Started streaming 138 records after 2 ms and completed after 10 ms.

6. Display a list of pairs of restaurants having more than three features in common.
Code:
Screenshot:

7. Display International restaurants that are open on Sunday.
Code:
MATCH (p:Places)--(r:Cuisines {cuisines: "International"})
MATCH (p:Places)--(o:OpeningHours {day:["Sun",""]})
return distinct p.place_id as id, p.place_name as name, r.cuisines as cuisine, o.day as day;
Screenshot:

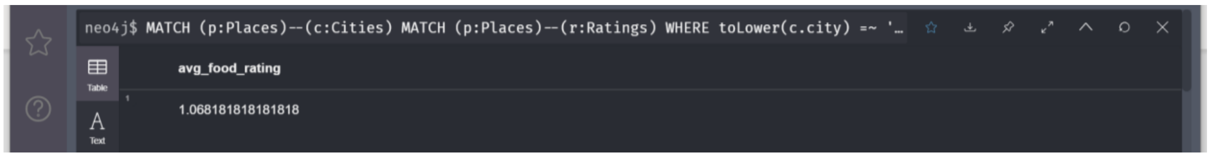


```
neo4j$ MATCH (p:Places)--(r:Cuisines {cuisines: "International"}) MATCH (p:Places)--(o:Openi…

   id       name                          cuisine            day
1  132862   "La Posada del Virrey"        "International"    ["Sun", ""]
2  134986   "Restaurant Las Mananitas"    "International"    ["Sun", ""]
3  132854   "Sirlone"                     "International"    ["Sun", ""]
4  132846   "el lechon potosino"          "International"    ["Sun", ""]

Started streaming 4 records after 1 ms and completed after 4 ms.
```

8. What is the average food rating for restaurants in city Victoria?
Code:

```
MATCH (p:Places)--(c:Cities)
MATCH (p:Places)--(r:Ratings)
WHERE toLower(c.city) =~ '.*victoria.*'
RETURN AVG(r.rating_food) as avg_food_rating;
```

Screenshot:

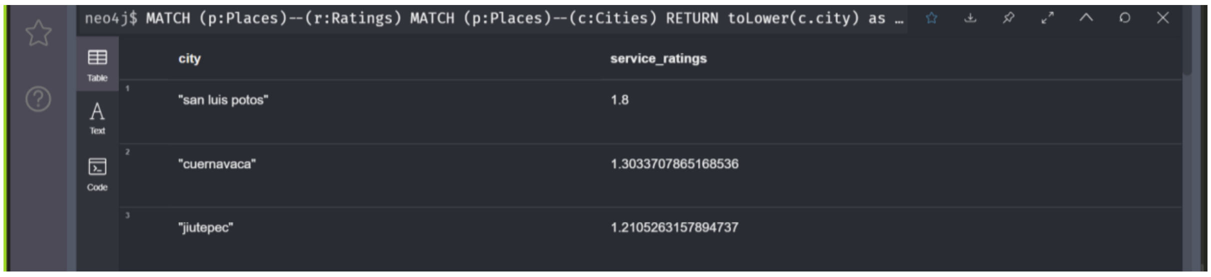


```
neo4j$ MATCH (p:Places)--(c:Cities) MATCH (p:Places)--(r:Ratings) WHERE toLower(c.city) =~ '…

   avg_food_rating
1  1.068181818181818
```

9. What are the top 3 most popular cities based on the total average service ratings?
Code:

```
MATCH (p:Places)--(r:Ratings)
MATCH (p:Places)--(c:Cities)
RETURN toLower(c.city) as city, AVG(r.rating_service) as service_ratings
ORDER BY service_ratings DESC
LIMIT 3;
```

Screenshot:

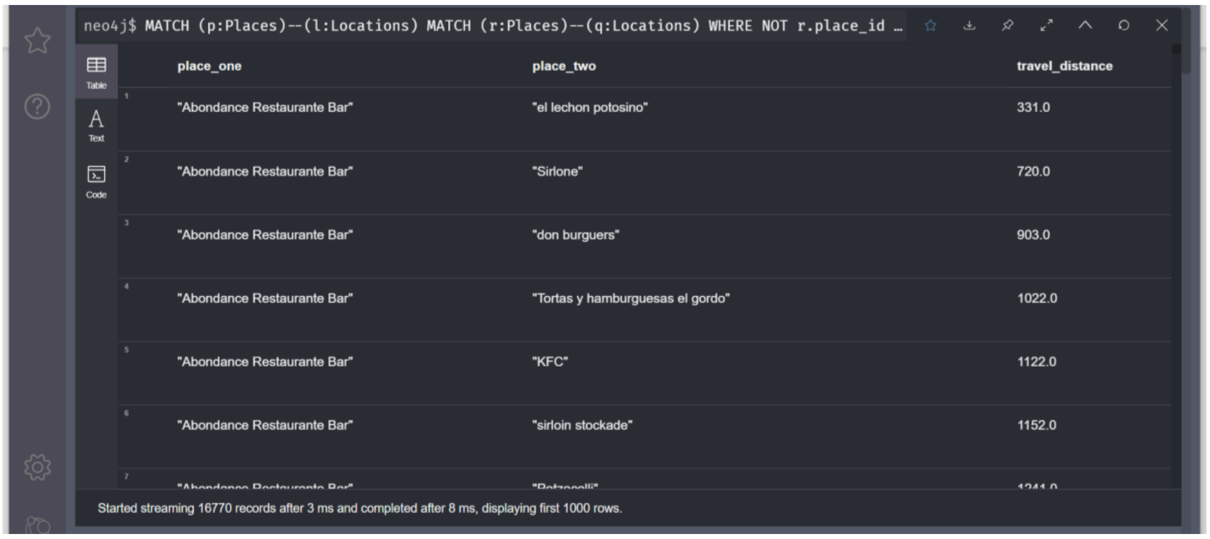


```
neo4j$ MATCH (p:Places)--(r:Ratings) MATCH (p:Places)--(c:Cities) RETURN toLower(c.city) as …

   city                 service_ratings
1  "san luis potos"     1.8
2  "cuernavaca"         1.3033707865168536
3  "jiutepec"           1.2105263157894737
```

10. For each place, rank other places that are close to each other by their locations. You will need to use the longitude and latitude to calculate the distance between places.
Code:

```
MATCH (p:Places)--(l:Locations)
MATCH (r:Places)--(q:Locations)
WHERE NOT r.place_id = p.place_id
WITH point({ longitude: l.longitude, latitude: l.latitude }) AS one_place,
point({ longitude: q.longitude, latitude: q.latitude }) AS second_place,
p.place_name AS place_one, r.place_name AS place_two
RETURN place_one, place_two, round(distance(one_place, second_place)) AS travel_distance
ORDER BY place_one,travel_distance;
```

Screenshot:



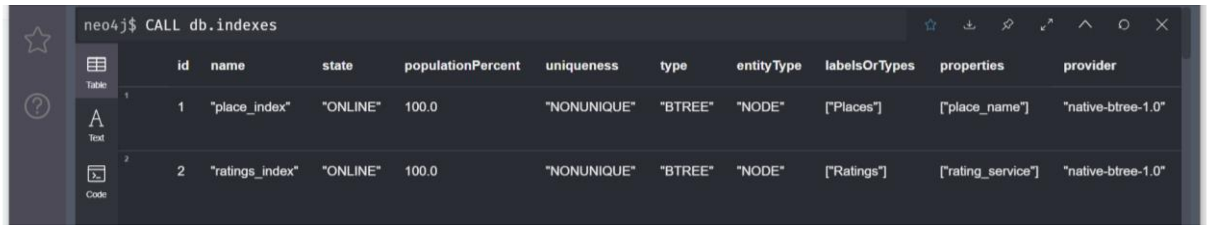
```
neo4j$ MATCH (p:Places)--(l:Locations) MATCH (r:Places)--(q:Locations) WHERE NOT r.place_id …
```

	place_one	place_two	travel_distance
1	"Abondance Restaurante Bar"	"el lechon potosino"	331.0
2	"Abondance Restaurante Bar"	"Sirlone"	720.0
3	"Abondance Restaurante Bar"	"don burguers"	903.0
4	"Abondance Restaurante Bar"	"Tortas y hamburguesas el gordo"	1022.0
5	"Abondance Restaurante Bar"	"KFC"	1122.0
6	"Abondance Restaurante Bar"	"sirloin stockade"	1152.0
7	"Abondance Restaurante Bar"	"Potzocalli"	1241.0

Started streaming 16770 records after 3 ms and completed after 8 ms, displaying first 1000 rows.

Two indexes have been created as shown below:
Code:
//index on place name
CREATE INDEX place_index FOR (p:Places)
ON (p.place_name);

//index on service ratings
CREATE INDEX ratings_index FOR (r:Ratings)
ON (r.rating_service);
Screenshot:



```
neo4j$ CALL db.indexes
```

	id	name	state	populationPercent	uniqueness	type	entityType	labelsOrTypes	properties	provider
1	1	"place_index"	"ONLINE"	100.0	"NONUNIQUE"	"BTREE"	"NODE"	["Places"]	["place_name"]	"native-btree-1.0"
2	2	"ratings_index"	"ONLINE"	100.0	"NONUNIQUE"	"BTREE"	"NODE"	["Ratings"]	["rating_service"]	"native-btree-1.0"

C.3 DATABASE MODIFICATIONS

1. MonR has gained some new information about a trendy new place

Code:
MERGE (p:Places {place_id:70000,place_name:"Taco Jacks"});

MATCH (p:Places {place_id:70000, place_name:"Taco Jacks"})
MATCH (s:Streets {street:"Carretera Central Sn"})
MERGE (p)-[:IS_ON]->(s);

MATCH (p:Places {place_id:70000, place_name:"Taco Jacks"})
MATCH (c:Cities {city:"San Luis Potosi"})
MERGE (p)-[:LOCATED_IN]->(c);

MATCH (p:Places {place_id:70000, place_name:"Taco Jacks"})
MATCH (st:States {state: "SLP"})
MERGE (p)-[:BELONGS_TO]->(st);

```
MATCH (p:Places {place_id:70000, place_name:"Taco Jacks"})
MATCH (co:Countries {country: "Mexico"})
MERGE (p)-[:PRESENT_IN]->(co);

MATCH (p:Places {place_id:70000, place_name:"Taco Jacks"})
MATCH (a:Alcohol {alcohol:"No_Alcohol_Served"})
MERGE (p)-[:ALCOHOL_IS]->(a);

MATCH (p:Places {place_id:70000, place_name:"Taco Jacks"})
MATCH (sm:Smoking {smoke: "not permitted"})
MERGE (p)-[:SMOKING_IS]->(sm);

MATCH (p:Places {place_id:70000, place_name:"Taco Jacks"})
MATCH (d:Dresscodes {dress_code:"informal"})
MERGE (p)-[:HAS_DRESSCODE]->(d);

MATCH (p:Places {place_id:70000, place_name:"Taco Jacks"})
MATCH (ac:Accessibility {accessibility:"completely"})
MERGE (p)-[:HAS]->(ac);

MATCH (p:Places {place_id:70000, place_name:"Taco Jacks"})
MATCH (pr:Prices {`price or budget`:"medium"})
MERGE (p)-[:HAS_PRICES]->(pr);

MATCH (p:Places {place_id:70000, place_name:"Taco Jacks"})
MATCH (f:Franchises {franchise:true})
MERGE (p)-[:HAS_FRANCHISE]->(f);

MATCH (p:Places {place_id:70000, place_name:"Taco Jacks"})
MATCH (ar:Area {area:"open"})
MERGE (p)-[:HAS_AREA]->(ar);

MATCH (p:Places {place_id:70000, place_name:"Taco Jacks"})
MATCH (o:Services {other_services: "Internet"})
MERGE (p)-[:HAS_SERVICES]->(o);

MATCH (p:Places {place_id:70000, place_name:"Taco Jacks"})
MATCH (pa:Parkings {parking:"none"})
MERGE (p)-[:HAS_PARKING]->(pa);

MATCH (p:Places {place_id:70000, place_name:"Taco Jacks"})
MATCH (pay:Payments {payment:"any"})
MERGE (p)-[:ACCEPTS]->(pay);

MATCH (p:Places {place_id:70000, place_name:"Taco Jacks"})
MATCH (cu:Cuisines {cuisines: "Mexican"})
MERGE (p)-[:SERVES]->(cu);

MATCH (p:Places {place_id:70000, place_name:"Taco Jacks"})
MATCH (cu:Cuisines {cuisines: "Burgers"})
```

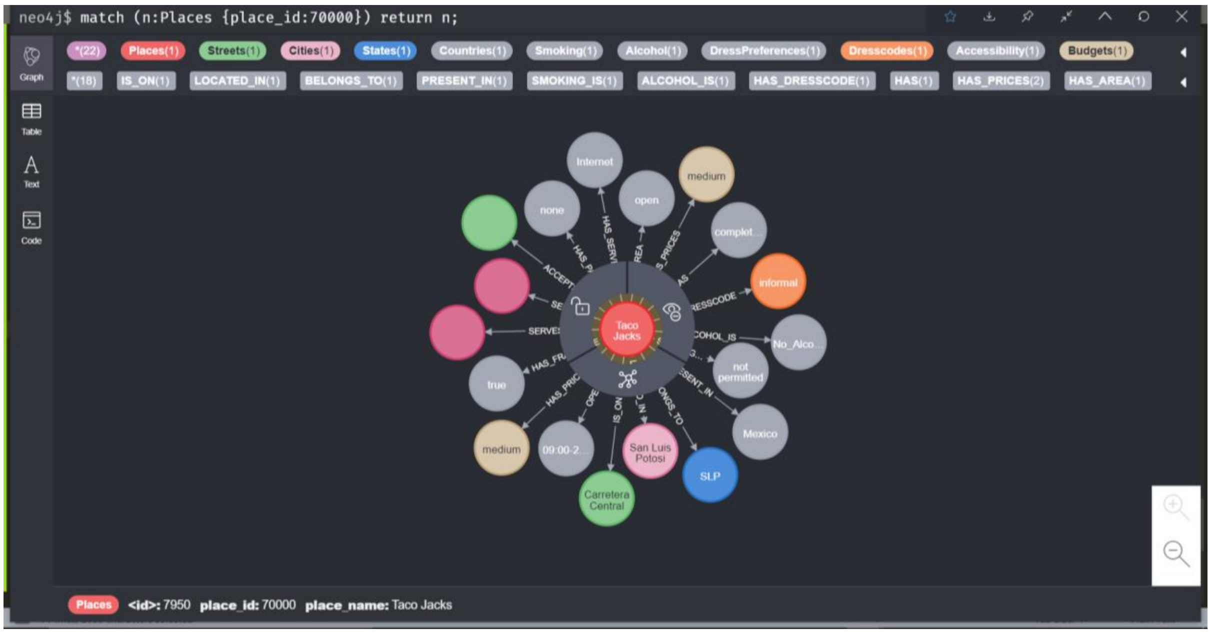
MERGE (p)-[:SERVES]->(cu);

MATCH (p:Places {place_id:70000, place_name:"Taco Jacks"})
MATCH (op:OpeningHours {hour:["09:00-20:00",""],day:["Mon","Tue","Wed","Thu","Fri",""]})
MERGE (p)-[:OPERATES]->(op);

MATCH (p:Places {place_id:70000, place_name:"Taco Jacks"})
MATCH (op:OpeningHours {hour:["12:00-18:00",""],day:["Sat",""]})
MERGE (p)-[:OPERATES]->(op);

MATCH (p:Places {place_id:70000, place_name:"Taco Jacks"})
MATCH (op:OpeningHours {hour:["12:00-18:00",""],day:["Sun",""]})
MERGE (p)-[:OPERATES]->(op);

Screenshot:



```
neo4j$ match (n:Places {place_id:70000}) return n;
```

Places <id>: 7950 place_id: 70000 place_name: Taco Jacks

*(22) Places(1) Streets(1) Cities(1) States(1) Countries(1) Smoking(1) Alcohol(1) DressPreferences(1) Dresscodes(1) Accessibility(1) Budgets(1)
*(18) IS_ON(1) LOCATED_IN(1) BELONGS_TO(1) PRESENT_IN(1) SMOKING_IS(1) ALCOHOL_IS(1) HAS_DRESSCODE(1) HAS(1) HAS_PRICES(2) HAS_AREA(1)
Graph
Table
Text
Code
Taco Jacks
Internet
medium
none
open
complet...
informal
No_Alco...
not permitted
Mexico
SLP
San Luis Potosi
Carretera Central
09:00-2...
medium
true

2. They have also realised that the user with user_id 1108, no longer prefers Fast_Food and also prefers to pay using debit_cards instead of cash. You are required to update user 1108's favorite cuisines and favorite payment methods.

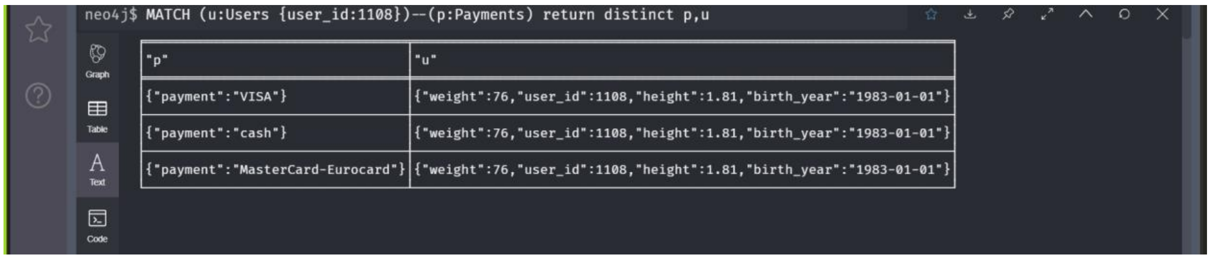
Code:

// delete relation with Fast_Food and cash
MATCH (u:Users {user_id:1108}) -[l:LOVES]-> (c:Cuisines {cuisines: "Fast_Food"}) DELETE l;
MATCH (u:Users {user_id:1108}) -[l:PAYS_BY]-> (c:Payments {payment: "cash"}) DELETE l;

//create a relationship for bank_debit_cards
MATCH (u:Users {user_id:1108}),(c:Payments {payment: "bank_debit_cards"})
CREATE (u)-[:PAYS_BY]->(c);

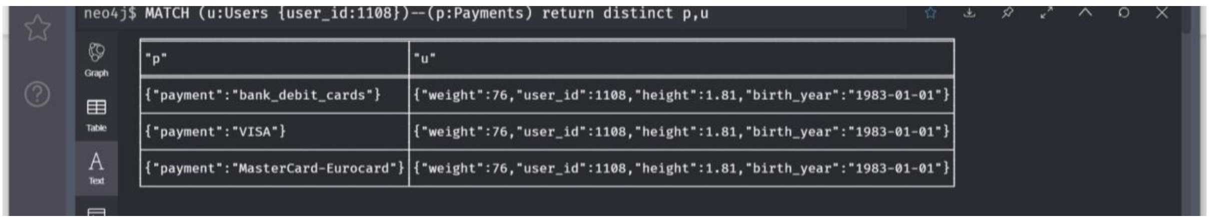
Screenshot:

Before:



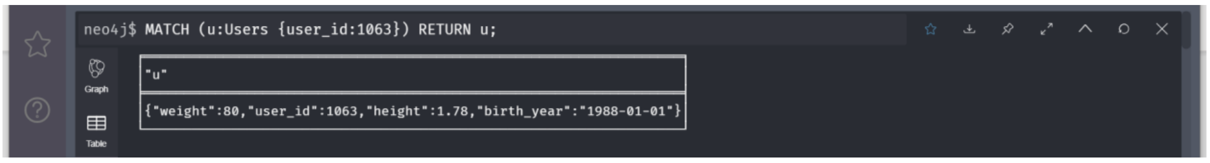
```
neo4j$ MATCH (u:Users {user_id:1108})--(p:Payments) return distinct p,u
"p"                               "u"
{"payment":"VISA"}                {"weight":76,"user_id":1108,"height":1.81,"birth_year":"1983-01-01"}
{"payment":"cash"}                {"weight":76,"user_id":1108,"height":1.81,"birth_year":"1983-01-01"}
{"payment":"MasterCard-Eurocard"} {"weight":76,"user_id":1108,"height":1.81,"birth_year":"1983-01-01"}
```

After:



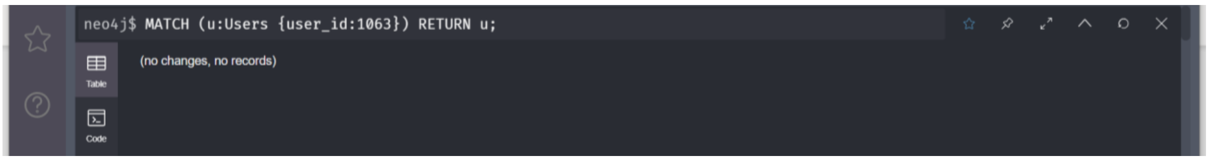
```
neo4j$ MATCH (u:Users {user_id:1108})--(p:Payments) return distinct p,u
"p"                               "u"
{"payment":"bank_debit_cards"}    {"weight":76,"user_id":1108,"height":1.81,"birth_year":"1983-01-01"}
{"payment":"VISA"}                {"weight":76,"user_id":1108,"height":1.81,"birth_year":"1983-01-01"}
{"payment":"MasterCard-Eurocard"} {"weight":76,"user_id":1108,"height":1.81,"birth_year":"1983-01-01"}
```

3. The management has realised that the user with user_id 1063 was an error. Therefore delete the user 1063 from the database.
Code:
MATCH (u:Users {user_id:1063}) DETACH DELETE u;
Screenshot:
Before:



```
neo4j$ MATCH (u:Users {user_id:1063}) RETURN u;
"u"
{"weight":80,"user_id":1063,"height":1.78,"birth_year":"1988-01-01"}
```

After:



```
neo4j$ MATCH (u:Users {user_id:1063}) RETURN u;
(no changes, no records)
```