**Task C.1 Database Design:**

At first I created new project called FIT5137_Assign2 and there I created a new graph called MonashBnBgraph. From the given data I identified nodes **Listing, Host** and **Review**. Host creates Listing and Review is for Listing. I created relationship between Host and Listing and between Review and Listing.To import data from CSV files and create relationships among them, I used following queries:

**For Host:**

```
LOAD CSV WITH HEADERS FROM "file:///host.csv"

AS row

WITH row WHERE row.host_id IS NOT NULL

MERGE (h:Host {host_id: row.host_id})

ON CREATE SET h.host_url = row.host_url,

h.host_name = row.host_name,

h.host_verifications = row.host_verifications,

h.host_since = row.host_since,

h.host_location = row.host_location,

h.host_response_time = row.host_response_time,

h.host_is_superhost = row.host_is_superhost
```

**For Listing:**

```
LOAD CSV WITH HEADERS FROM "file:///listing.csv"

AS row

WITH row WHERE row.id IS NOT NULL

MERGE (l:Listing {id: row.id})

ON CREATE SET l.name = row.name,

l.summary = row.summary,

l.listing_url = row.listing_url,

l.picture_url = row.picture_url,

l.host_id = row.host_id,

l.neighbourhood = row.neighbourhood,
```

```
l.street = row.street,
l.zipcode = row.zipcode,
l.latitude = row.latitude,
l.longitude = row.longitude,
l.room_type = row.room_type,
l.amenities = row.amenities,
l.price = row.price,
l.extra_people = row.extra_people,
l.minimum_nights = row.minimum_nights,
l.calculated_host_listings_count = row.calculated_host_listings_count,
l.availability_365 = row.availability_365
```

**For Review:**

```
LOAD CSV WITH HEADERS FROM "file:///review.csv"
AS row
WITH row WHERE row.id IS NOT NULL
MERGE (r:Review {review_id: row.id})
ON CREATE SET
r.listing_id = row.listing_id,
r.reviewer_name = row.reviewer_name,
r.review_scores_rating = row.review_scores_rating,
r.reviewer_id = row.reviewer_id,
r.date=row.date
```

**Create Relationships**

**From Host to Listing**

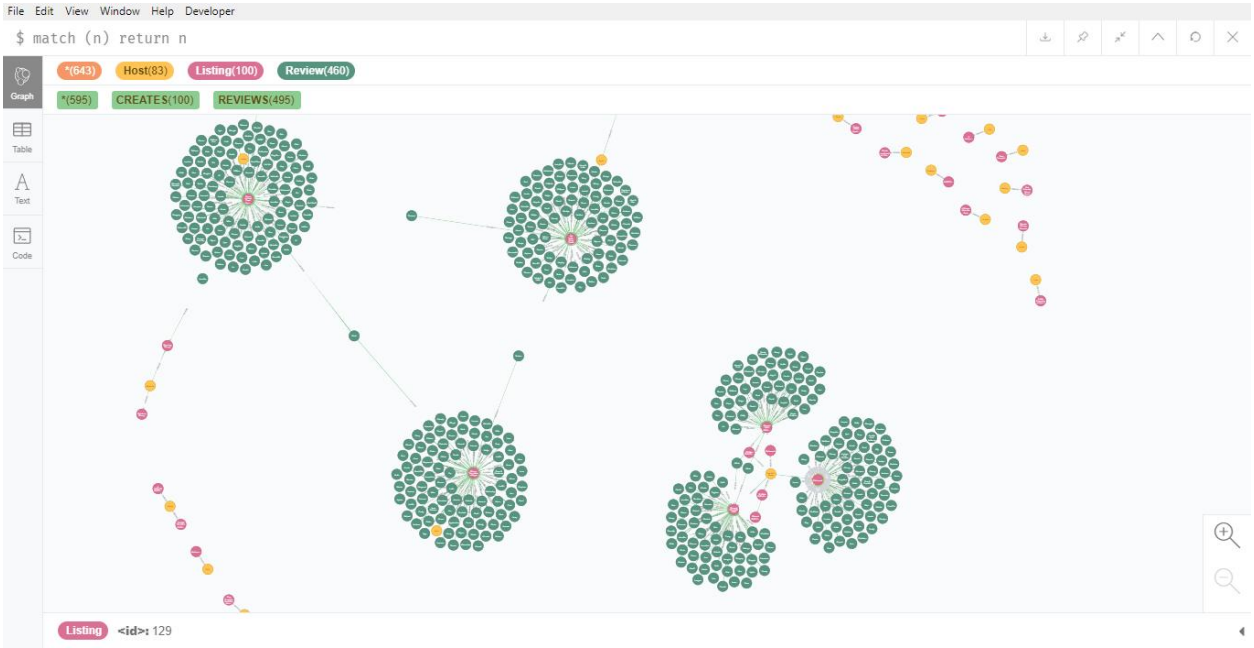
```
LOAD CSV WITH HEADERS FROM "file:///host.csv" AS csvLine
```

```
MATCH (l:Listing {host_id: csvLine.host_id})
MATCH (h:Host {host_id: csvLine.host_id})
CREATE (h)-[c:CREATES]->(l)
return h,l,c
```

**From Review to Listing**

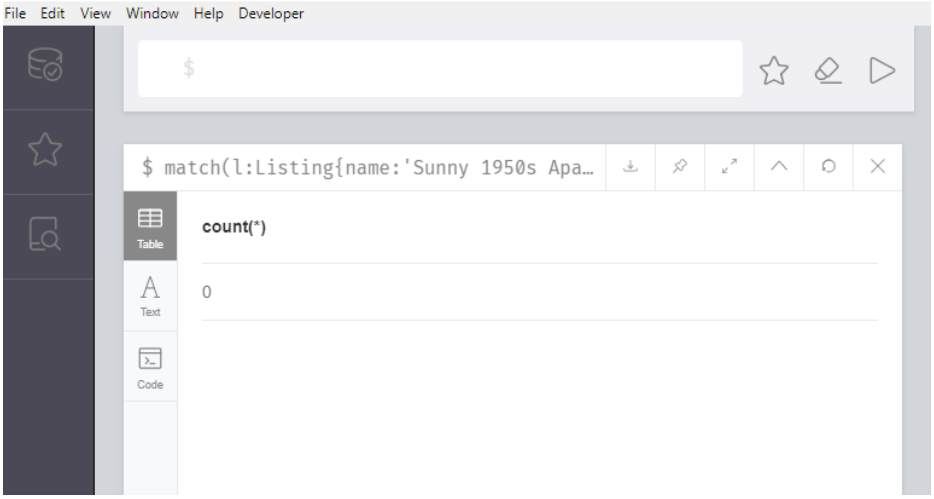
```
LOAD CSV WITH HEADERS FROM "file:///review.csv" AS csvLine
MATCH (l:Listing {id: csvLine.listing_id})
MATCH (r:Review {listing_id: csvLine.listing_id})
CREATE (r)-[b:REVIEWS]->(l)
return l,b,r
```

Output:

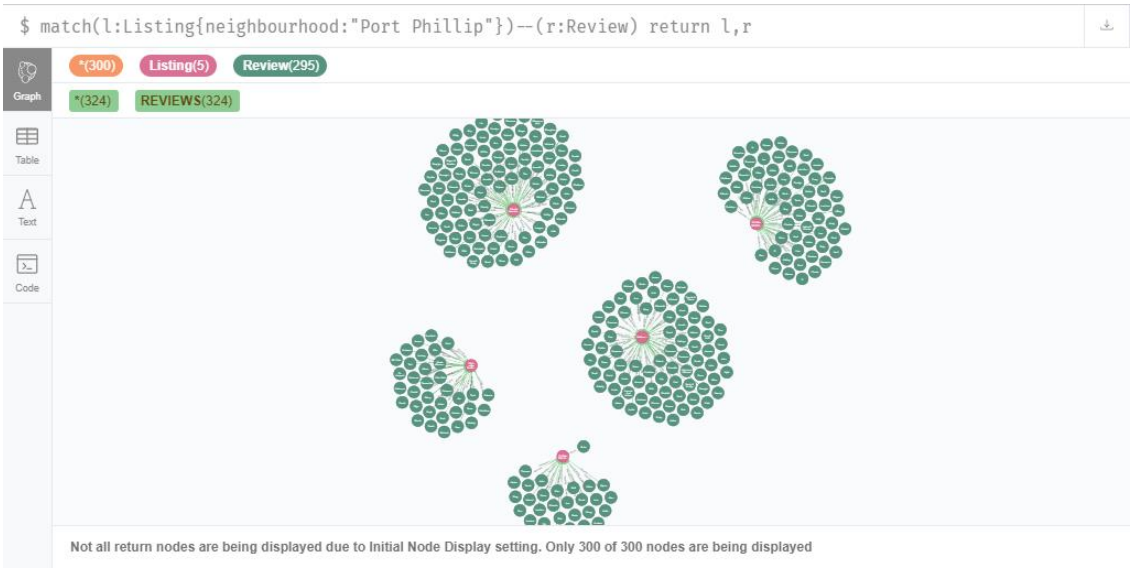



**Task C.2 Queries**

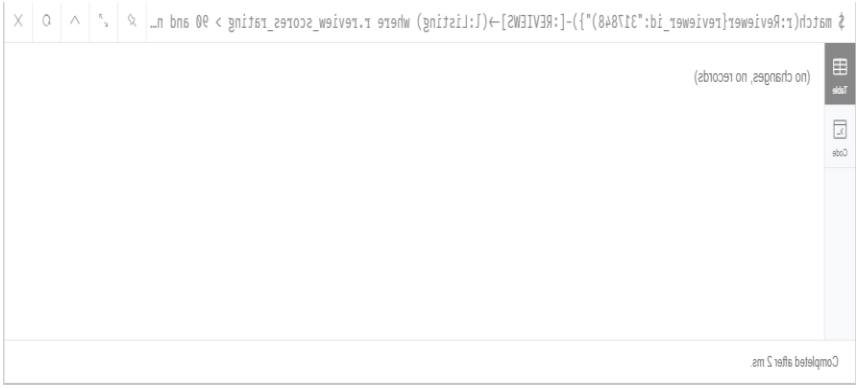
1. match(l:Listing{name:'Sunny 1950s Apartment, St Kilda East Longer stays'})
   return count(*)




2. match(l:Listing{neighbourhood:"Port Phillip"})--(r:Review)
   return l,r



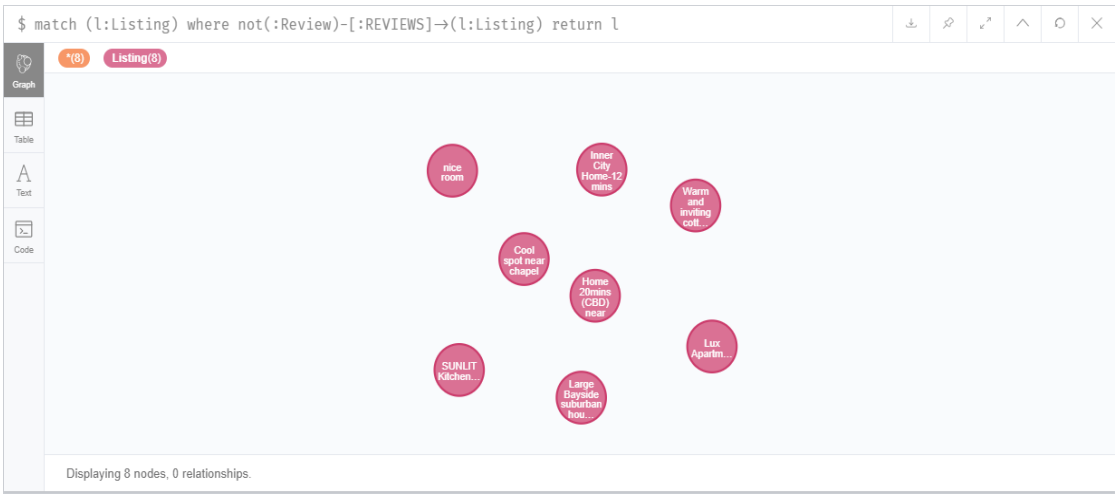

3. match(r:Reviewer{reviewer_id:"317848)"})-[:REVIEWS]->(l:Listing)
   where r.review_scores_rating > 90
   and not exists ((r:Reviewer{reviewer_id:"4162110"})-[:REVIEWS]->(l:Listing))
   return r,l



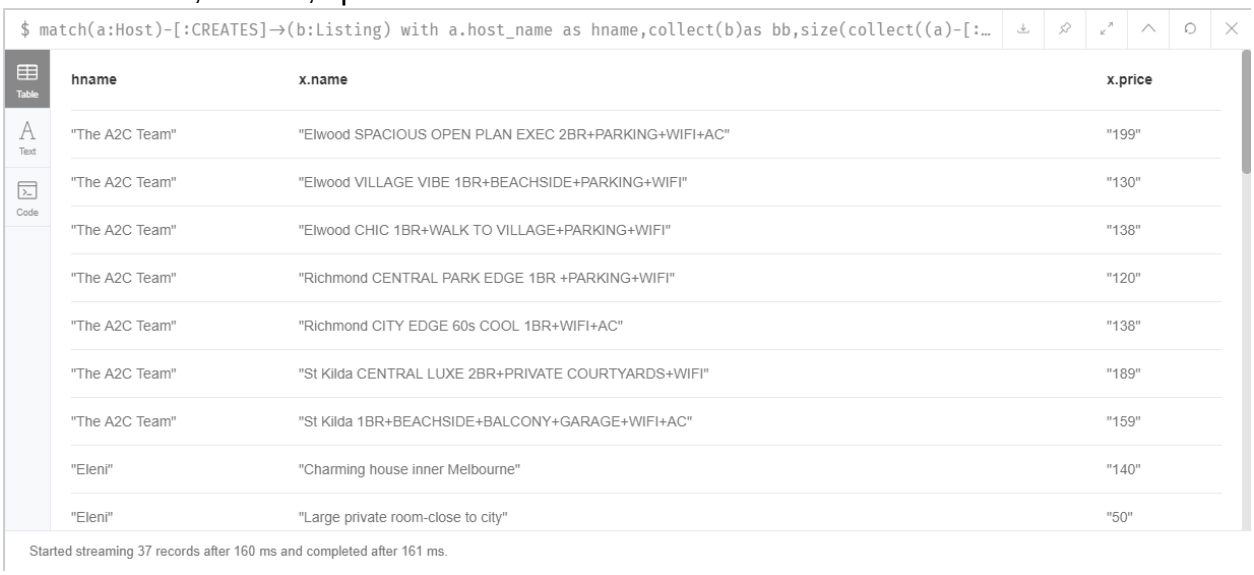

4. Match (l:Listing)
   where not l.amenities Contains "Wifi"
   return l.name,l.street


5. match(r:Review)-[re:REVIEWS]->(l:Listing)
   return count(re)


6. blank
7. match (l:Listing)
   where not(:Review)-[:REVIEWS]->(l:Listing)
   return l




8. match(a:Host)-[:CREATES]->(b:Listing)
   with a.host_name as hname,collect(b)as bb,size(collect((a)-[:CREATES]->(b)))as lc
   where lc>1
   unwind bb as x
   return hname,x.name,x.price




| hname | x.name | x.price |
|---|---|---|
| "The A2C Team" | "Elwood SPACIOUS OPEN PLAN EXEC 2BR+PARKING+WIFI+AC" | "199" |
| "The A2C Team" | "Elwood VILLAGE VIBE 1BR+BEACHSIDE+PARKING+WIFI" | "130" |
| "The A2C Team" | "Elwood CHIC 1BR+WALK TO VILLAGE+PARKING+WIFI" | "138" |
| "The A2C Team" | "Richmond CENTRAL PARK EDGE 1BR +PARKING+WIFI" | "120" |
| "The A2C Team" | "Richmond CITY EDGE 60s COOL 1BR+WIFI+AC" | "138" |
| "The A2C Team" | "St Kilda CENTRAL LUXE 2BR+PRIVATE COURTYARDS+WIFI" | "189" |
| "The A2C Team" | "St Kilda 1BR+BEACHSIDE+BALCONY+GARAGE+WIFI+AC" | "159" |
| "Eleni" | "Charming house inner Melbourne" | "140" |
| "Eleni" | "Large private room-close to city" | "50" |

9. match(l:Listing{neighbourhood:"Melbourne"})
   return avg(toInt(l.price))

```
$ match(l:Listing{neighbourhood:"Melbourne"}) return avg(toInt(l.price))
```

Table

| avg(toInt(l.price)) |
| --- |
| 176.34999999999997 |

Text

Code

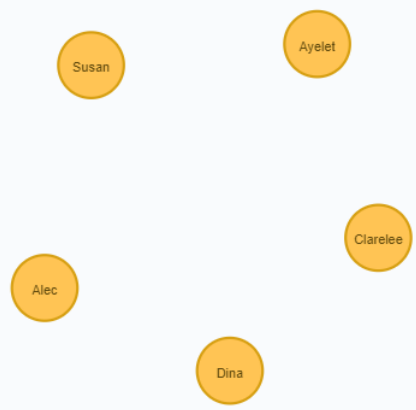
Started streaming 1 records after 16 ms and completed after 16 ms.

10. match(h:Host)-[:CREATES]->(l:Listing)
    return toINT(l.price),l.street,h
    order by toINT(l.price) desc
    limit 5

```
$ match(h:Host)-[:CREATES]→(l:Listing) return toINT(l.price),l.street,h order by toINT(l.price) desc…
```

Graph

*(5) Host(5)




Displaying 5 nodes, 0 relationships.

11. match(a:Review)-[:REVIEWS]->(b:Listing)
    where substring(a.date,0,4)="2017"
    return count(b.name)

```
$ match(a:Review)-[:REVIEWS]→(b:Listing) where substring(a.date,0,4)="2017" return count(b.name)
```

| count(b.name) |
|---|
| 1349 |

12. match(r:Review)-[:REVIEWS]->(l:Listing)
    return avg(toInt(r.review_scores_rating)), l.neighbourhood
    order by avg(toInt(r.review_scores_rating)) desc
    limit 10

```
$ match(r:Review)-[:REVIEWS]→(l:Listing) return avg(toInt(r.review_scores_rating)), l.neighbourhood …
```

| avg(toInt(r.review_scores_rating)) | l.neighbourhood |
|---|---|
| 98.11111111111111 | "Banyule" |
| 95.55643044619428 | "Boroondara" |
| 95.2727272727273 | "Bayside" |
| 95.25630252100834 | "Kingston" |
| 95.05380333951766 | "Stonnington" |
| 95.05263157894734 | "Casey" |
| 94.90539682539683 | "Melbourne" |
| 94.85714285714289 | "Brimbank" |
| 94.80263157894738 | "Hobsons Bay" |

Started streaming 10 records after 98 ms and completed after 99 ms.

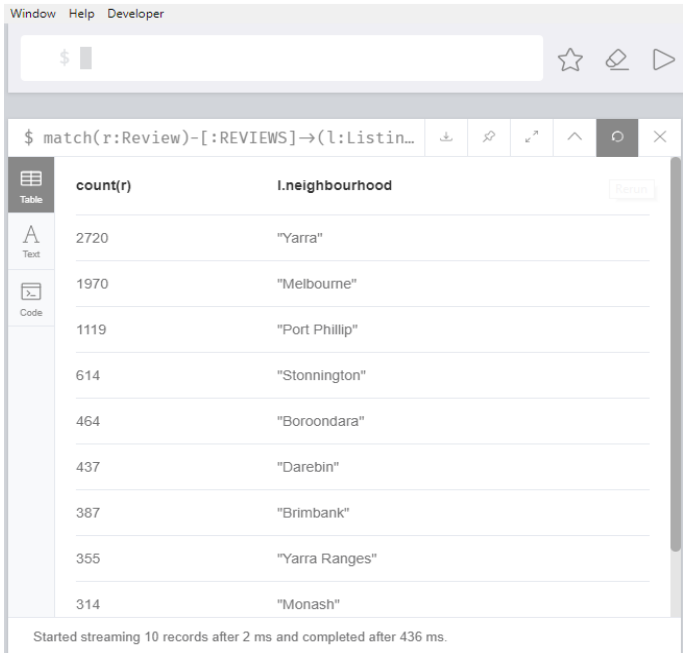
13. match(h:Host)-[:CREATES]->(l:Listing)
    where h.host_location <> l.street
    return h.host_name,h.host_location,l.name,l.street

14. match(l:Listing)
    with (toString(l.extra_people)*2)*5 as people,(toString(l.price)*2)*5 as tprice,l.name as name,l.street as address,l.price as price_per_night,l.extra_people as extra_price
    return name,address,price_per_night,extra_price,tprice
    order by tprice

**Additional 5 Queries:**

1. List top 10 neighbourhoods according to number of reviews.

```
match(r:Review)-[:REVIEWS]->(l:Listing)
return count(r),l.neighbourhood
order by count(r) desc
limit 10
```




2. List top 10 host name who got highest number of reviews.

```
match(r:Review)-[:REVIEWS]->(l:Listing)
match(h:Host)-[:CREATES]->(l:Listing)
return count(r),h.host_name
order by count(r) desc
limit 10
```

$ match(r:Review)-[:REVIEWS]→(l:Listin…

Table | Text | Code

| count(r) | h.host_name |
|---|---|
| 1280 | "Ramona" |
| 435 | "Karen" |
| 423 | "Little Journey" |
| 421 | "Mario" |
| 387 | "Ryan" |
| 385 | "The A2C Team" |
| 293 | "Rebecca" |
| 290 | "Judy" |
| 272 | "Dina" |

Started streaming 10 records after 2 ms and completed after 98 ms.

**3.** List top 10 reviewer name accoding to the number of reviews they provided.

```
match(r:Review)-[:REVIEWS]->(l:Listing)
with r.reviewer_name as rname,collect(l)as ll,size(collect((r)-[:REVIEWS]->(l))) as lc
return rname, lc
order by lc desc
limit 10
```

$ match(r:Review)-[:REVIEWS]→(l:Listin…

| rname | lc |
|---|---|
| "John" | 103 |
| "David" | 94 |
| "Michael" | 86 |
| "Tania" | 84 |
| "Andrew" | 70 |
| "Chris" | 68 |
| "Paul" | 64 |
| "Sarah" | 63 |
| "Peter" | 56 |

Started streaming 10 records after 149 ms and completed after 149 ms.

4. List the listings which have availability between 300 and 200. And create a descending order.

```
match (l:Listing)
where toINT(l.availability_365)>200 and toINT(l.availability_365)<300
return l.name,l.street,toINT(l.availability_365)
order by toINT(l.availability_365) desc
```

```
$ match (l:Listing) where toINT(l.avail…
```

| l.name | l.street | toINT(l.availability_365) |
|---|---|---|
| "Melbourne 2 Bedrooms 2 Bathrooms FULL Kitchen" | "Hadfield, VIC, Australia" | 297 |
| "City Location-Perfect for Singles" | "Melbourne, VIC, Australia" | 287 |
| "Melbourne BnB near City & Sports" | "St Kilda East, VIC, Australia" | 274 |
| "Collingwood 2 bedrm Warehouse Apt" | "Collingwood, VIC, Australia" | 265 |
| "City's edge Penthouse - private bath, views!" | "South Melbourne, VIC, Australia" | 261 |
| "Sunny 1950s Apartment, St Kilda | "Saint Kilda East, | 259 |

Started streaming 11 records after 18 ms and completed after 18 ms.

5. How many private rooms in the listing?

```
match(l:Listing)
where l.room_type="Private room"
return count(l.room_type)
```

```
$ match(l:Listing) where l.room_type="P…
```

| count(l.room_type) |
|---|
| 50 |

Started streaming 1 records after 2 ms and completed after 3 ms.

**Index:**

1. Index on neighbourhoods.

create index on:Listing(neighbourhood);

I used it because in few queries, I've used neighbourhood.

2. Index on price and amenities.

create index on:Listing(price,amenities);

3. Index on review date, reviewer name and ratings.

create index on:revie(date,reviewer_name,review_scores_rating);

The reason for using index on price,amenities,date,reviewer name and ratings is they have been used several times in my queries.So, it would be easier to find them using index.

**Task C.3 Database Modifications.**

1. From AirBnB website, I added three new listings and their corresponding hosts details and reviews. After that I created relationships among them. I used following queries to add and create relationships.
   **Creating listing**
   create (l:Listing{listing_id:1001,
   name:"Unique Cob Cottage",
   summary:"Appearing in numerous books on natural building, our cottage is a welcoming and cozy retreat hand sculpted of local.",
   listing_url:"https://www.airbnb.com.au/rooms/1720832",
   picture_url:"https://www.airbnb.com.au/rooms/1720832?source_impression_id=p3_1571622270_LBlb2HzwCOWXQK%2BK",
   host_id:9071324,
   neighbourhood:"Vancouver",
   street:"Mayne Island,BC,Canada",
   zipcode:250,
   latitude:48.6667,
   longitude:-123.95,
   room_type:"Entire home",
   amenities:"{WiFi,Heating,Hot water,Iron,Refrigerator,BBQ grill,Smoke detector}",
   price:177,
   extra_people:50,
   minimum_nights:2,
   availability_365:150})

   create (l:Listing{listing_id:1002,
   name:"The World Famous Seashell House",
   summary:"The world famous Seashell house is a gated property.",

listing_url:"https://www.airbnb.com.au/rooms/530250",
picture_url:"https://www.airbnb.com.au/rooms/530250?source_impression_id=p3_1571617870_4XNQY%2BwhIy4WEvoU",
host_id:481799,
neighbourhood:"La Gloria",
street:"Isla Mujeres,Mexico",
zipcode:77400,
latitude:21.2321696,
longitude:-86.7667381,
room_type:"Entire home",
amenities:"{WiFi,PayTV,Hot water,Iron,Refrigerator,Air conditioning,BBQ grill,Hair dryer,Smoke detector}",
price:441,
extra_people:80,
minimum_nights:3,
availability_365:100})

create (l:Listing{listing_id:1003,
name:"I SETTE CONI TRULLO EDERA",
summary:"Spend a unforgettable holiday in the enchanting surroundings of the town of Cisternino",
listing_url:"https://www.airbnb.com.au/rooms/432044",
picture_url:"https://www.airbnb.com.au/rooms/432044?source_impression_id=p3_1571617506_9Lq60kVX%2F7D7Igi%2B",
host_id:294274,
neighbourhood:"Campanile",
street:"Ostuni,BR,Italy",
zipcode:72017,
latitude:40.7268956,
longitude:17.562942,
room_type:"Entire home",
amenities:"{WiFi,Heating,Hot water,Cot,Refrigerator,Air conditioning,Kitchen,Smoke detector}",
price:114,
extra_people:20,
minimum_nights:1,
availability_365:180})

**creating host**
create(h:host{
host_id:9071324,
host_url:"https://www.airbnb.com.au/users/show/9071324",
host_name:"Alexis",

host_verifications:"['email', 'phone', 'facebook', 'government_id']",
host_since:2013-12-06,
host_location:"British Columbia,Canada",
host_response_time:"within a day",
host_is_superhost:true
})

create(h:host{
host_id:481799,
host_url:"https://www.airbnb.com.au/users/show/481799",
host_name:"Michelle",
host_verifications:['email', 'phone', 'government_id'],
host_since:2011-11-02,
host_location:"Cancun,Mexico",
host_response_time:"within an hour",
host_is_superhost:false
})

create(h:host{
host_id:294274,
host_url:"https://www.airbnb.com.au/users/show/294274",
host_name:"Anna",
host_verifications:['email', 'phone', 'reviews'],
host_since:2010-06-03,
host_location:"Ostuni, Italy",
host_response_time:"within an hour",
host_is_superhost:false
})

**Creating reviews**

create(r:review{
listing_id:1001,
id:575993,
date:2014-02-28,
reviewer_id:494947,
reviewer_name:"Hilary",
review_scores_rating:93,
comments:"Very hospitable, much appreciated."
})

create(r:review{
listing_id:1001,
id:755759,

```
date:2014-06-15,
reviewer_id:35355,
reviewer_name:"Melissa",
review_scores_rating:91,
comments:"This was my first time using airbnb and it was great."
})

create(r:review{
listing_id:1002,
id:35633,
date:2012-07-11,
reviewer_id:23456,
reviewer_name:"Michale",
review_scores_rating:94,
comments:"Breakfast was a pleasant bonus"
})

create(r:review{
listing_id:1002,
id:53536,
date:2015-11-19,
reviewer_id:54454,
reviewer_name:"Stuart",
review_scores_rating:90,
comments:"Spent some lovely time"
})

create(r:review{
listing_id:1002,
id:45535,
date:2013-03-21,
reviewer_id:65666,
reviewer_name:"Petra",
review_scores_rating:92,
comments:"Spent some lovely time"
})
```

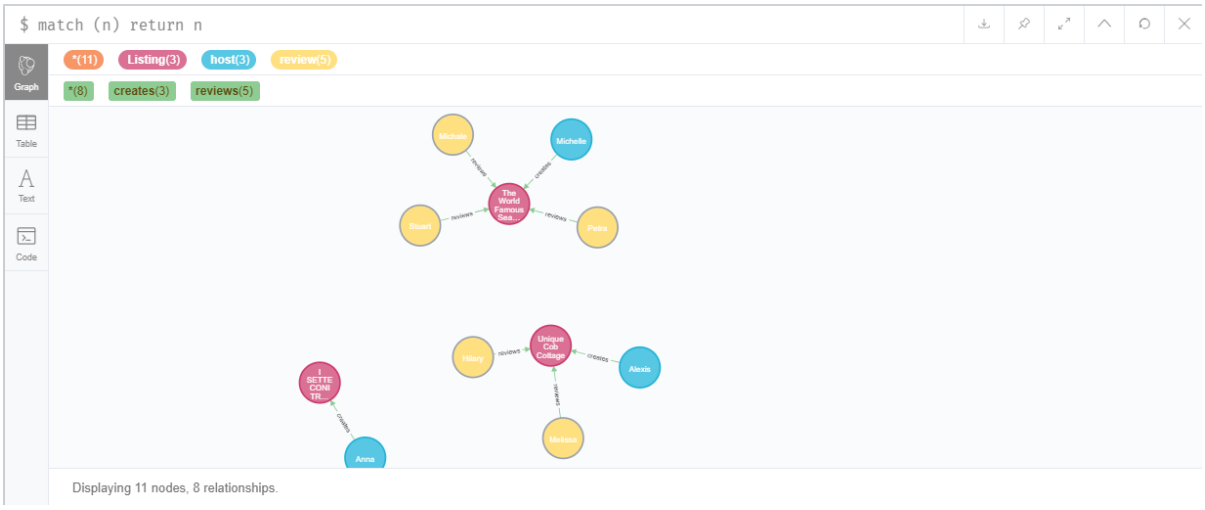
**Create Relationships:**
**From host to listing:**

```
match(h:host{host_name:"Alexis"})
match(l:Listing{name:"Unique Cob Cottage"})
create (h)-[c1:creates]->(l)
return h,c1,l
```

```
match(h:host{host_name:"Michelle"})
match(l:Listing{name:"The World Famous Seashell House"})
create (h)-[c2:creates]->(l)
return h,c2,l

match(h:host{host_name:"Anna"})
match(l:Listing{name:"I SETTE CONI TRULLO EDERA"})
create (h)-[c3:creates]->(l)
return h,c3,l
```

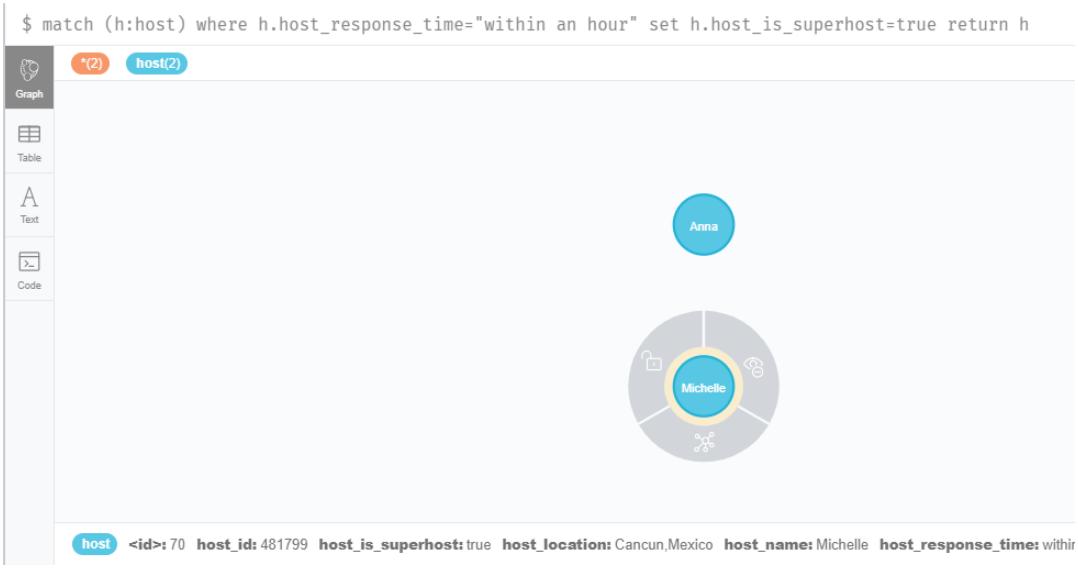
**From review to listing:**

```
match(r:review{reviewer_name:"Hilary"})
match(l:Listing{name:"Unique Cob Cottage"})
create (r)-[re1:reviews]->(l)
return r,re1,l

match(r:review{reviewer_name:"Melissa"})
match(l:Listing{name:"Unique Cob Cottage"})
create (r)-[re2:reviews]->(l)
return r,re2,l

match(r:review{reviewer_name:"Michale"})
match(l:Listing{name:"The World Famous Seashell House"})
create (r)-[re3:reviews]->(l)
return r,re3,l

match(r:review{reviewer_name:"Stuart"})
match(l:Listing{name:"The World Famous Seashell House"})
create (r)-[re4:reviews]->(l)
return r,re4,l

match(r:review{reviewer_name:"Petra"})
match(l:Listing{name:"The World Famous Seashell House"})
create (r)-[re5:reviews]->(l)
return r,re5,l
```




2. match(h:host)
   where toString(h.host_since)="2009"
   set h.host_verifications="Facebook"
   return h


3. match (h:host)
   where h.host_response_time="within an hour"
   set h.host_is_superhost=true
   return h



4. 
```
match(r:review)
match(l:Listing)
match(h:host)
match(r)-[re:reviews]->(l)
where not toString(r.date)>"2017"
create (h:host{active:false})
return h
```

5. 
```
match(l:Listing{availability_365:0})
where not (:review)-[:reviews]->(l:listing)
detach delete l
```

**C.4. Advanced Topic.**

**Option 2:**

Here I've considered AirBnB example. As the workers of Airbnb expanded rapidly throughout the world and their information environment became increasingly difficult to navigate, profitability was hindered. They are looking for Neo4j to help in creating a simple, user-friendly master data management system known as the Dataportal.

Airbnb connects people with different travel experiences for leasing or renting short-term accommodation with an online sharing platform. Using a number of search filters, a user can quickly crawl through the site's more than 4 million listings of rentals that span 65,000 cities and 191 countries. Airbnb reported sales of over $1 billion for the third quarter of 2018 and is valued at $31 billion.

In a large and complex organization, the stored data is also large. So, they can become unmanageable and restrictive as they are distributed in different platform. Before using Neo4j, airbnb relied on 200,000 tables in their main Hive data warehouse spread across multiple clusters,10,000 Superset charts and dashboards, 6,000 experiments in metrics, over 6,000 Tableau workbooks and charts, and over 1,500 knowledge posts. And employees relied on tribal knowledge for answers to questions, which ultimately choked productivity.

Then the developer team realized that using neo4j is the best option for their ecosystem and they applied it. According to Bodley, a software enginner working at airbnb, said that they have used neo4j for four main reason. Firstly, its logical. Secondly, it's nimble. Thirdly, it's popular and last of all it can integrate very well. Using neo4j, we can search through millions of data connections faster that using any other DBMS.

Airbnb uses Elasticsearch and Python. Neo4j can integrate airbnb's preferred language very well and allows them to enrich search rankings by taking advantage of search topology. Everyday, a huge amount of data are being pushed into neo4j graph database from Hive in order to facilitate quick and highly relevant contextual search results. From

Hive, the flow moves into two directions. The nodes are pushed into Elasticsearch via a transaction hook-based GraphAware plugin. From there, Elasticsearch serves as search engine. After that, they use Flask as a lightweight Python web app. It is used with other data tools. Web servers uses the obtained Results from Elasticsearch.

Some Suggestions to improve current graph database of MonashBnB:

1. Some rich data set can be used.
2. While writing queries, cypher could be used carefully.
3. To avoid hassles, a cluster must get running.