FIT5137 – ADVANCE DATABASE TECHNOLOGY:

ASSIGNMENT: 2

Student Id: 29825660

Student Name: Arvind Yogesh Attur Ramesh

ASSESSMENT COVER SHEET

Student ID number	*29825660*
Given Name	*ARVIND YOGESH*
Family name	*ATTUR RAMESH*

Unit Name and Code:	**FIT5137 – Advance Database Technology**		
Campus:	**Caulfield**		
Assignment Title:	**MonashBnB**		
Name of Lecturer:	**Agnes Haryanto**		
Name of Tutor:	**Agnes Haryanto**		
Tutorial Day and Time:	**Tuesday: 12-2 pm**		
Phone Number:	**0415974649**		
Email Address:	**aatt0002@student.monash.edu**		
Has any part of this assignment been previously submitted as part of another unit/course? ☐ Yes ☐ No			
Due Date:	**25/10/2019**	Date Submitted:	**23/10/2019**

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 The material used can be from any source (staff, students or the internet, published and unpublished works).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:

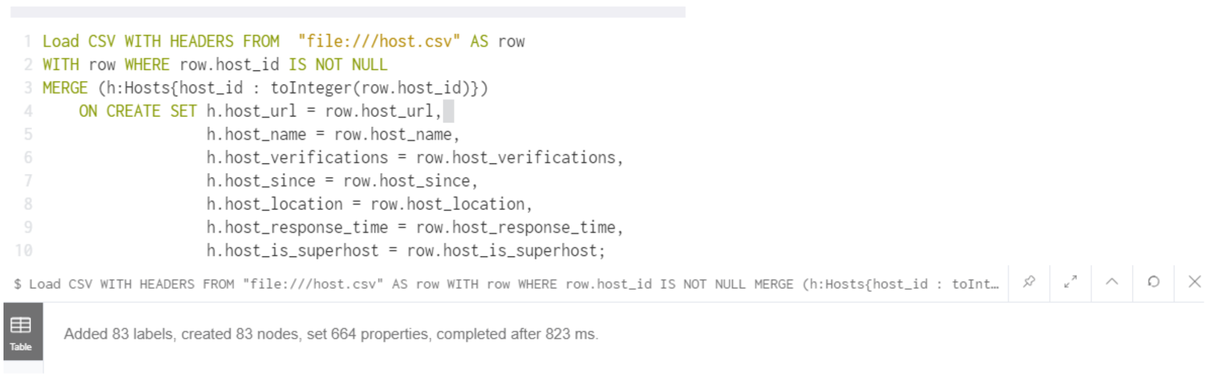
- I have read the university's Student Academic Integrity Policy and Procedures.
- I understand the consequences of engaging in plagiarism and collusion as described in Part 7 of the Monash University (Council) Regulations http://adm.monash.edu/legal/legislation/statutes
- have taken proper care to safeguard 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 ...***Arvind Yogesh AR***................. Date: ***23/10/2019***.......................................

* delete (iii) if not applicable

Screenshots for all the Queries:

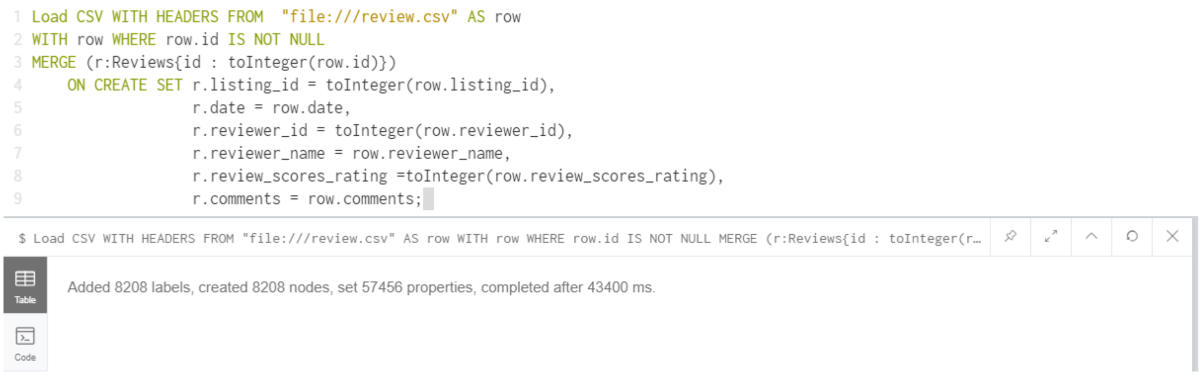
Task C1: Import CSV files:



```
Load CSV WITH HEADERS FROM  "file:///host.csv" AS row
WITH row WHERE row.host_id IS NOT NULL
MERGE (h:Hosts{host_id : toInteger(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;
```

```
$ Load CSV WITH HEADERS FROM "file:///host.csv" AS row WITH row WHERE row.host_id IS NOT NULL MERGE (h:Hosts{host_id : toInt…
```

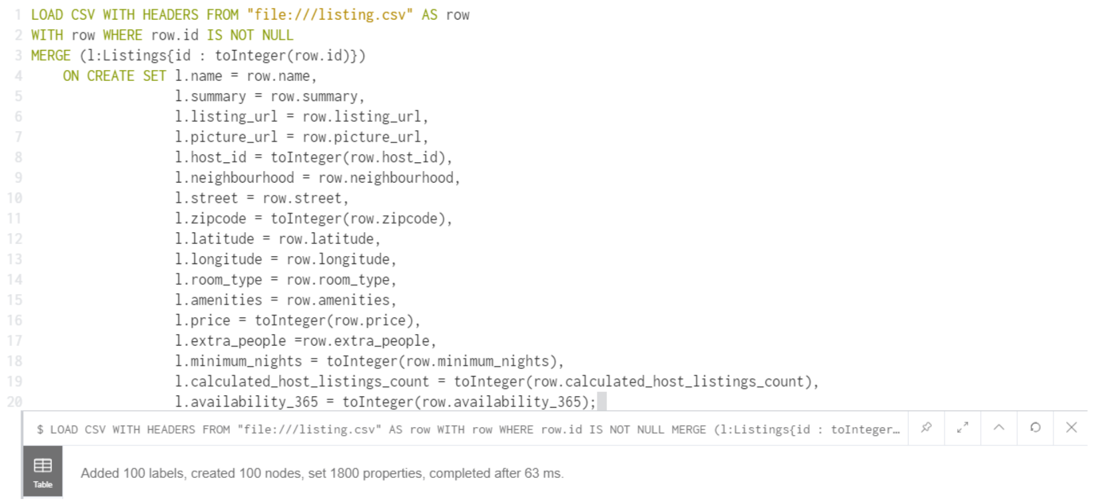
Added 83 labels, created 83 nodes, set 664 properties, completed after 823 ms.



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

```
$ Load CSV WITH HEADERS FROM "file:///review.csv" AS row WITH row WHERE row.id IS NOT NULL MERGE (r:Reviews{id : toInteger(r…
```

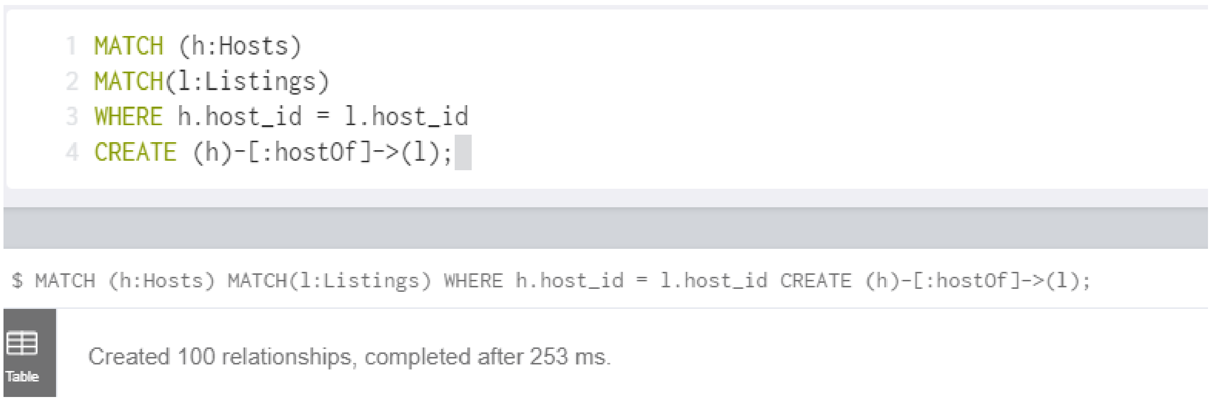
Added 8208 labels, created 8208 nodes, set 57456 properties, completed after 43400 ms.



```
LOAD CSV WITH HEADERS FROM "file:///listing.csv" AS row
WITH row WHERE row.id IS NOT NULL
MERGE (l:Listings{id : toInteger(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 = toInteger(row.host_id),
                  l.neighbourhood = row.neighbourhood,
                  l.street = row.street,
                  l.zipcode = toInteger(row.zipcode),
                  l.latitude = row.latitude,
                  l.longitude = row.longitude,
                  l.room_type = row.room_type,
                  l.amenities = row.amenities,
                  l.price = toInteger(row.price),
                  l.extra_people =row.extra_people,
                  l.minimum_nights = toInteger(row.minimum_nights),
                  l.calculated_host_listings_count = toInteger(row.calculated_host_listings_count),
                  l.availability_365 = toInteger(row.availability_365);
```

```
$ LOAD CSV WITH HEADERS FROM "file:///listing.csv" AS row WITH row WHERE row.id IS NOT NULL MERGE (l:Listings{id : toInteger…
```

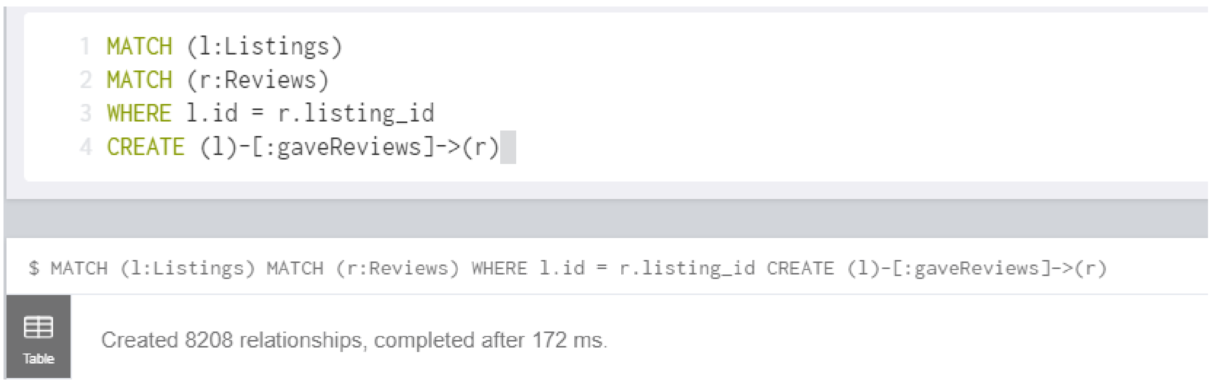
Added 100 labels, created 100 nodes, set 1800 properties, completed after 63 ms.



```
MATCH (h:Hosts)
MATCH(l:Listings)
WHERE h.host_id = l.host_id
CREATE (h)-[:hostOf]->(l);
```

```
$ MATCH (h:Hosts) MATCH(l:Listings) WHERE h.host_id = l.host_id CREATE (h)-[:hostOf]->(l);
```

```
Created 100 relationships, completed after 253 ms.
```



```
MATCH (l:Listings)
MATCH (r:Reviews)
WHERE l.id = r.listing_id
CREATE (l)-[:gaveReviews]->(r)
```

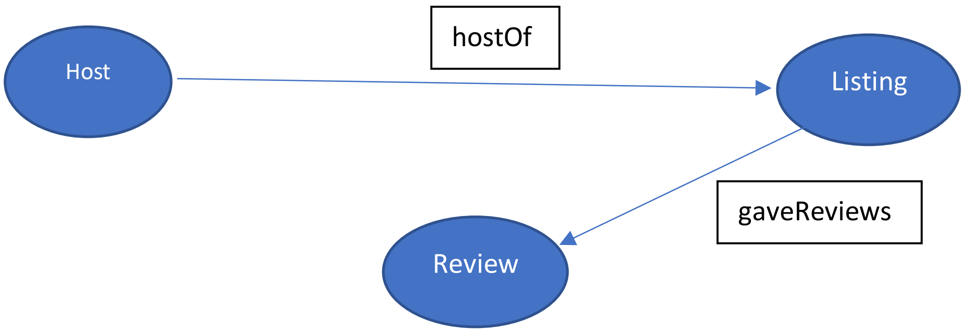
```
$ MATCH (l:Listings) MATCH (r:Reviews) WHERE l.id = r.listing_id CREATE (l)-[:gaveReviews]->(r)
```

```
Created 8208 relationships, completed after 172 ms.
```

This design is approached as it is easy and most of the domain-based queries could be answered using this design. AsNeo4j graph database accepts only integer, float and string at the time of import. Properties like "host_id", "listing_id" and few others are converted to integer using "ToInteger". Other properties are imported as "string" datatype by default. In this design, three node and 2 relationships are identified. Hosts, Listings and Reviews are the nodes and relationships are

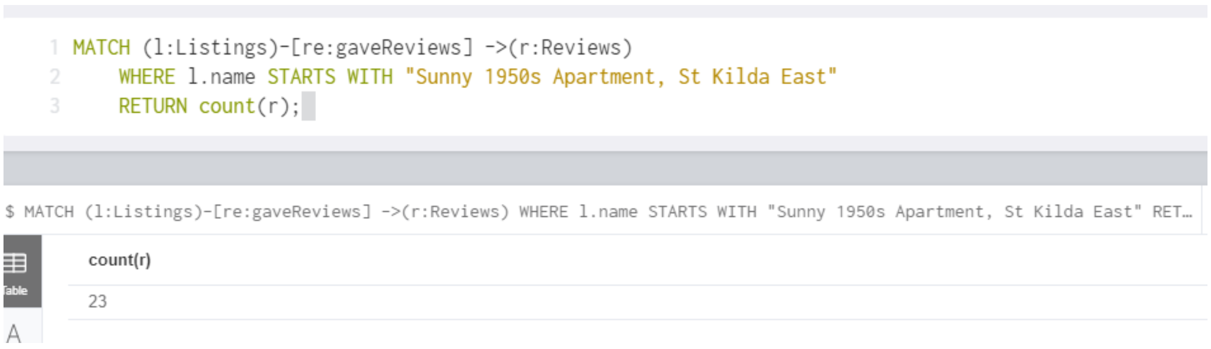
1) "hostOf" relationship is created between hosts and listing nodes using "host_id" property as it is common in both nodes.
2) "gaveReviews" relationship is created between listing and review nodes using "listing_id" property as it is common in both nodes.

<u>Graph Illustration:</u>



Task C2:

1) How many reviews does “Sunny 1950s Apartment, St Kilda East” have?

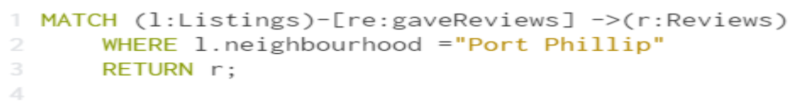


```
MATCH (l:Listings)-[re:gaveReviews] ->(r:Reviews)
   WHERE l.name STARTS WITH "Sunny 1950s Apartment, St Kilda East"
   RETURN count(r);
```

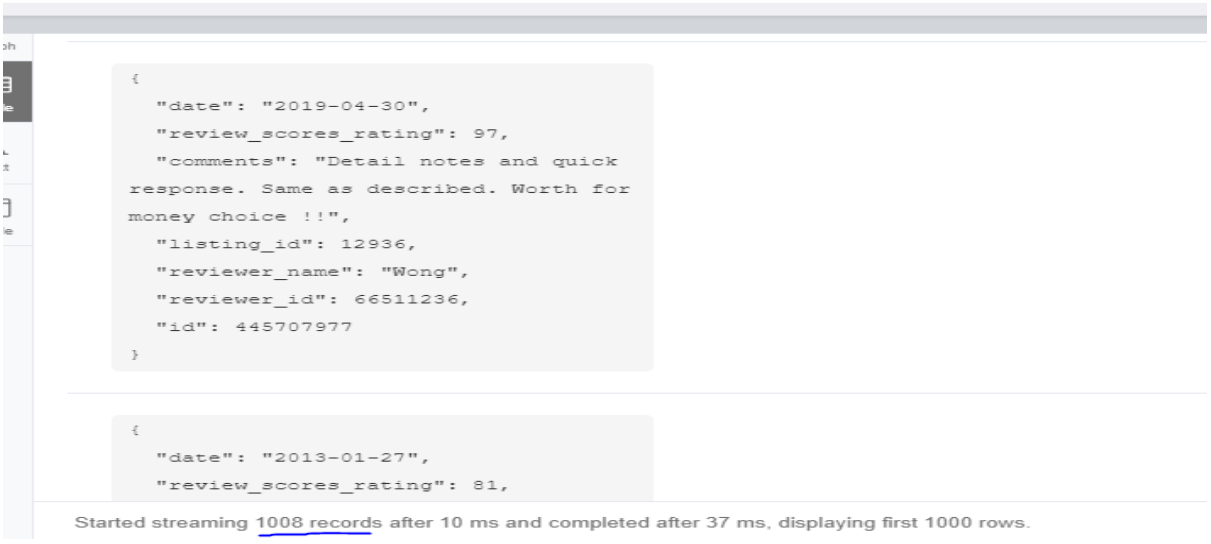
```
$ MATCH (l:Listings)-[re:gaveReviews] ->(r:Reviews) WHERE l.name STARTS WITH "Sunny 1950s Apartment, St Kilda East" RET…
```

count(r)
23

2) Show all reviews in Port Phillip.



```
MATCH (l:Listings)-[re:gaveReviews] ->(r:Reviews)
    WHERE l.neighbourhood ="Port Phillip"
    RETURN r;
```

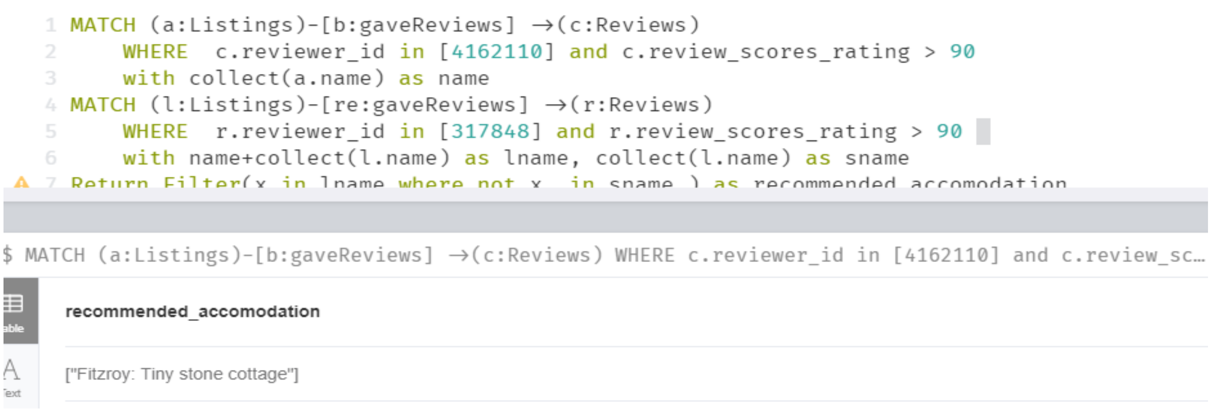


```
{
  "date": "2019-04-30",
  "review_scores_rating": 97,
  "comments": "Detail notes and quick
response. Same as described. Worth for
money choice !!",
  "listing_id": 12936,
  "reviewer_name": "Wong",
  "reviewer_id": 66511236,
  "id": 445707977
}
```

```
{
  "date": "2013-01-27",
  "review_scores_rating": 81,
```

Started streaming 1008 records after 10 ms and completed after 37 ms, displaying first 1000 rows.

3) Can you recommend accommodations that Jerome (reviewer 4162110) has never been but Sandy & Pete (reviewer 317848) have stayed and gave ratings above 90?



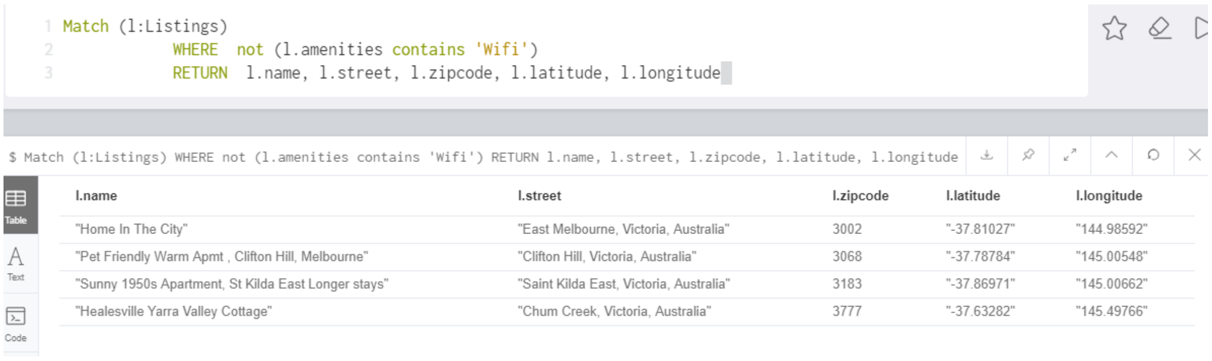
```
MATCH (a:Listings)-[b:gaveReviews] →(c:Reviews)
    WHERE  c.reviewer_id in [4162110] and c.review_scores_rating > 90
    with collect(a.name) as name
MATCH (l:Listings)-[re:gaveReviews] →(r:Reviews)
    WHERE  r.reviewer_id in [317848] and r.review_scores_rating > 90
    with name+collect(l.name) as lname, collect(l.name) as sname
Return Filter(x in lname where not x  in sname ) as recommended_accomodation
```

```
$ MATCH (a:Listings)-[b:gaveReviews] →(c:Reviews) WHERE c.reviewer_id in [4162110] and c.review_sc…
```

recommended_accomodation
["Fitzroy: Tiny stone cottage"]

As reviewer “Sandy & Pete” used “Private room” accommodation which was already used by “Jerome”, no unused accommodation can be recommended.

4) List all accommodation names and locations that do not provide Wi-Fi.

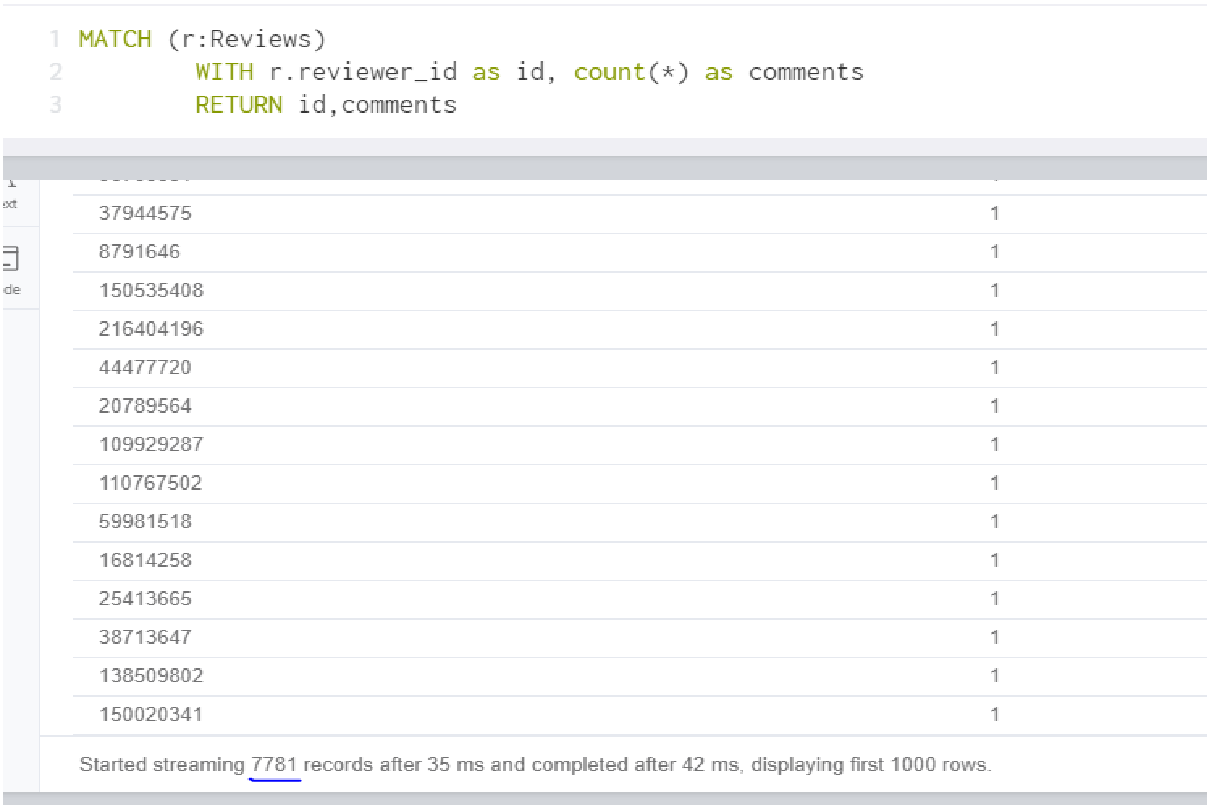


```
Match (l:Listings)
        WHERE  not (l.amenities contains 'Wifi')
        RETURN  l.name, l.street, l.zipcode, l.latitude, l.longitude
```

```
$ Match (l:Listings) WHERE not (l.amenities contains 'Wifi') RETURN l.name, l.street, l.zipcode, l.latitude, l.longitude
```

l.name	l.street	l.zipcode	l.latitude	l.longitude
"Home In The City"	"East Melbourne, Victoria, Australia"	3002	"-37.81027"	"144.98592"
"Pet Friendly Warm Apmt , Clifton Hill, Melbourne"	"Clifton Hill, Victoria, Australia"	3068	"-37.78784"	"145.00548"
"Sunny 1950s Apartment, St Kilda East Longer stays"	"Saint Kilda East, Victoria, Australia"	3183	"-37.86971"	"145.00662"
"Healesville Yarra Valley Cottage"	"Chum Creek, Victoria, Australia"	3777	"-37.63282"	"145.49766"

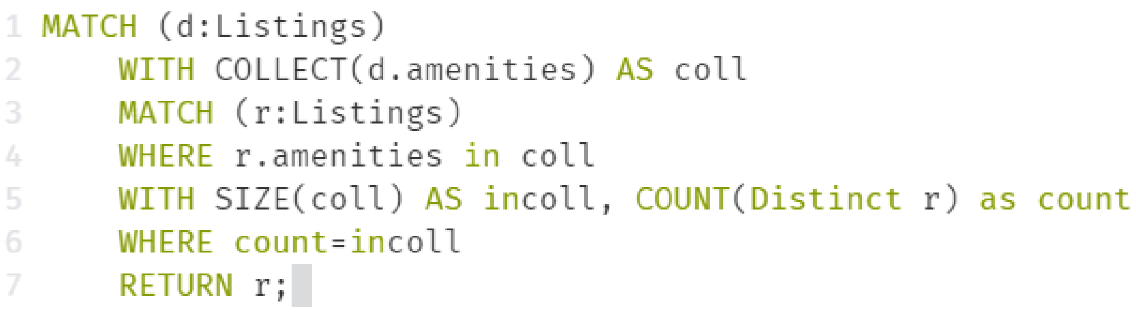
5) Count how many times a reviewer left reviews.



```
MATCH (r:Reviews)
        WITH r.reviewer_id as id, count(*) as comments
        RETURN id,comments
```

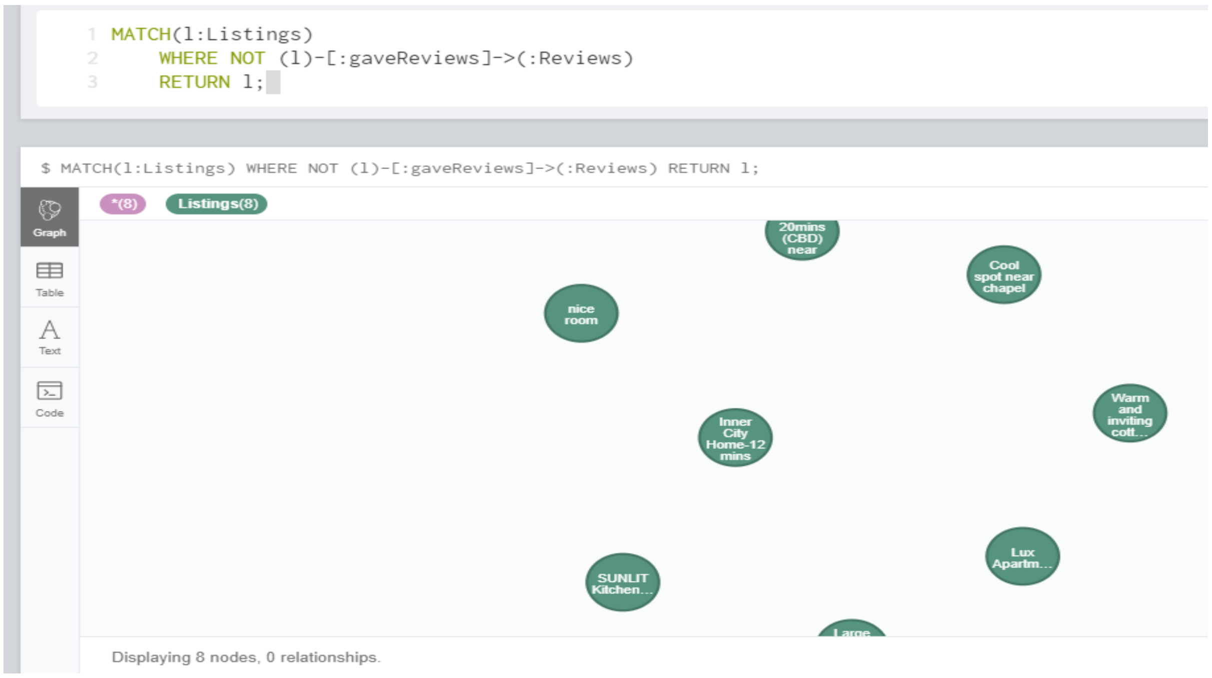
```
37944575                                                        1
8791646                                                         1
150535408                                                       1
216404196                                                       1
44477720                                                        1
20789564                                                        1
109929287                                                       1
110767502                                                       1
59981518                                                        1
16814258                                                        1
25413665                                                        1
38713647                                                        1
138509802                                                       1
150020341                                                       1

Started streaming 7781 records after 35 ms and completed after 42 ms, displaying first 1000 rows.
```

6) Display a list of pairs of accommodations having more than three amenities in common.



```
MATCH (d:Listings)
    WITH COLLECT(d.amenities) AS coll
    MATCH (r:Listings)
    WHERE r.amenities in coll
    WITH SIZE(coll) AS incoll, COUNT(Distinct r) as count
    WHERE count=incoll
    RETURN r;
```

7) Which listings do not have any review?

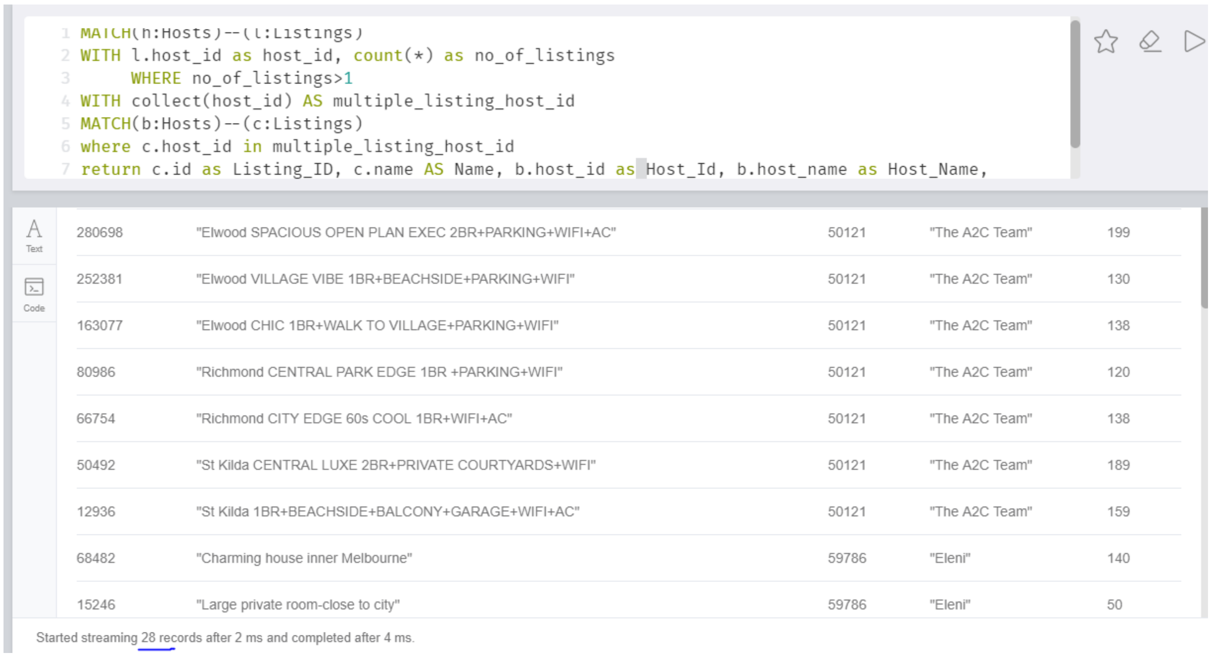


```
MATCH(l:Listings)
    WHERE NOT (l)-[:gaveReviews]->(:Reviews)
    RETURN l;
```

```
$ MATCH(l:Listings) WHERE NOT (l)-[:gaveReviews]->(:Reviews) RETURN l;
```

Displaying 8 nodes, 0 relationships.

8) Show all hosts who have multiple listings. Display both the host details and the listing name and price.

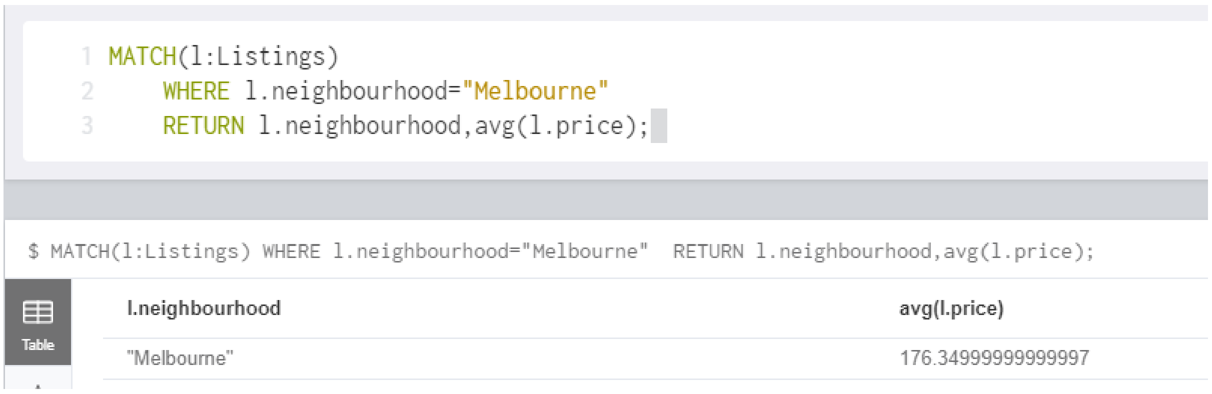


```
MATCH(h:Hosts)--(l:Listings)
WITH l.host_id as host_id, count(*) as no_of_listings
    WHERE no_of_listings>1
WITH collect(host_id) AS multiple_listing_host_id
MATCH(b:Hosts)--(c:Listings)
where c.host_id in multiple_listing_host_id
return c.id as Listing_ID, c.name AS Name, b.host_id as Host_Id, b.host_name as Host_Name,
```

280698	"Elwood SPACIOUS OPEN PLAN EXEC 2BR+PARKING+WIFI+AC"	50121	"The A2C Team"	199
252381	"Elwood VILLAGE VIBE 1BR+BEACHSIDE+PARKING+WIFI"	50121	"The A2C Team"	130
163077	"Elwood CHIC 1BR+WALK TO VILLAGE+PARKING+WIFI"	50121	"The A2C Team"	138
80986	"Richmond CENTRAL PARK EDGE 1BR +PARKING+WIFI"	50121	"The A2C Team"	120
66754	"Richmond CITY EDGE 60s COOL 1BR+WIFI+AC"	50121	"The A2C Team"	138
50492	"St Kilda CENTRAL LUXE 2BR+PRIVATE COURTYARDS+WIFI"	50121	"The A2C Team"	189
12936	"St Kilda 1BR+BEACHSIDE+BALCONY+GARAGE+WIFI+AC"	50121	"The A2C Team"	159
68482	"Charming house inner Melbourne"	59786	"Eleni"	140
15246	"Large private room-close to city"	59786	"Eleni"	50

Started streaming 28 records after 2 ms and completed after 4 ms.

9) What is the average price for accommodations in Melbourne neighbourhood?



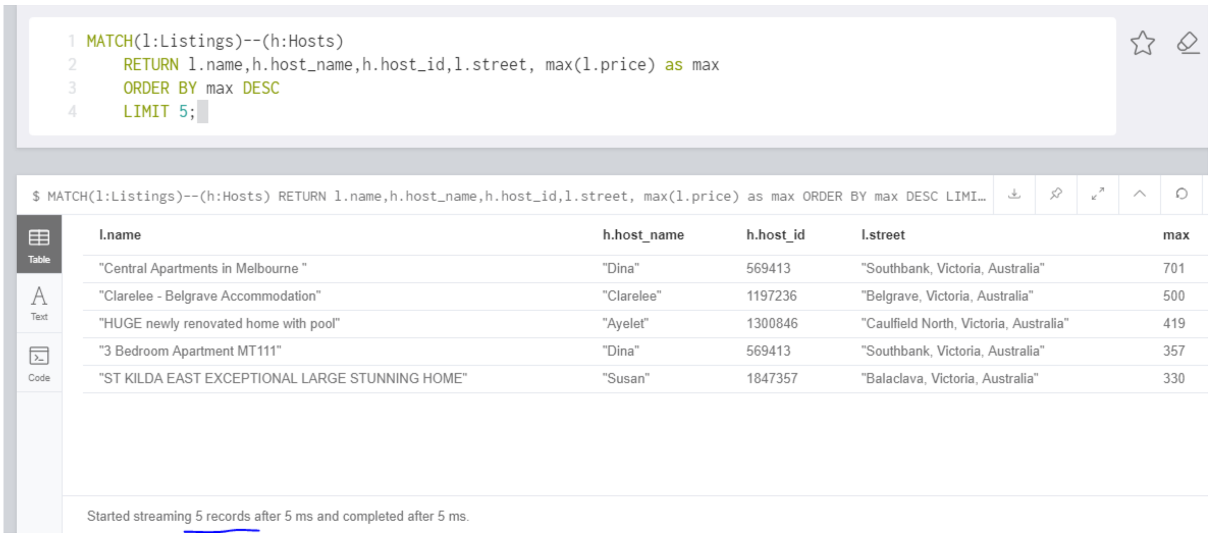
```
MATCH(l:Listings)
    WHERE l.neighbourhood="Melbourne"
    RETURN l.neighbourhood,avg(l.price);
```

```
$ MATCH(l:Listings) WHERE l.neighbourhood="Melbourne"  RETURN l.neighbourhood,avg(l.price);
```

Table

l.neighbourhood	avg(l.price)
"Melbourne"	176.34999999999997

10) Where are the top 5 most expensive accommodations? Display the locations, host information, and names of those accommodation.



```
MATCH(l:Listings)--(h:Hosts)
    RETURN l.name,h.host_name,h.host_id,l.street, max(l.price) as max
    ORDER BY max DESC
    LIMIT 5;
```

```
$ MATCH(l:Listings)--(h:Hosts) RETURN l.name,h.host_name,h.host_id,l.street, max(l.price) as max ORDER BY max DESC LIMI…
```

Table

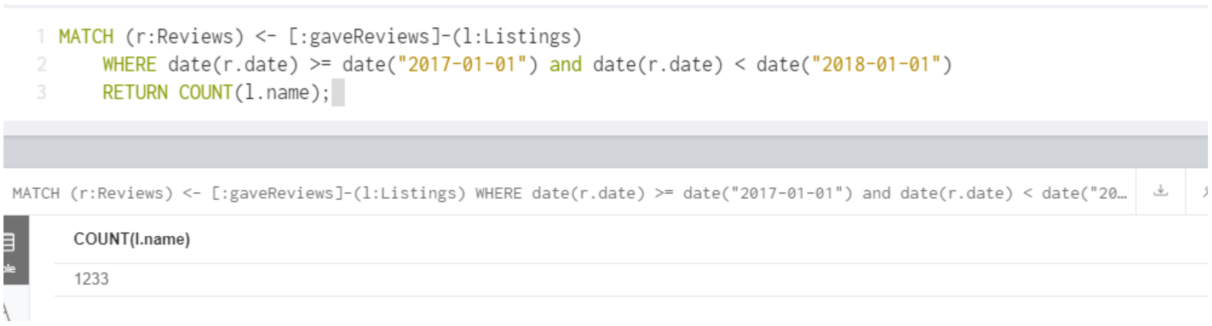
l.name	h.host_name	h.host_id	l.street	max
"Central Apartments in Melbourne "	"Dina"	569413	"Southbank, Victoria, Australia"	701
"Clarelee - Belgrave Accommodation"	"Clarelee"	1197236	"Belgrave, Victoria, Australia"	500
"HUGE newly renovated home with pool"	"Ayelet"	1300846	"Caulfield North, Victoria, Australia"	419
"3 Bedroom Apartment MT111"	"Dina"	569413	"Southbank, Victoria, Australia"	357
"ST KILDA EAST EXCEPTIONAL LARGE STUNNING HOME"	"Susan"	1847357	"Balaclava, Victoria, Australia"	330

Text

Code

Started streaming 5 records after 5 ms and completed after 5 ms.

11) How many accommodations were reviewed in 2017?

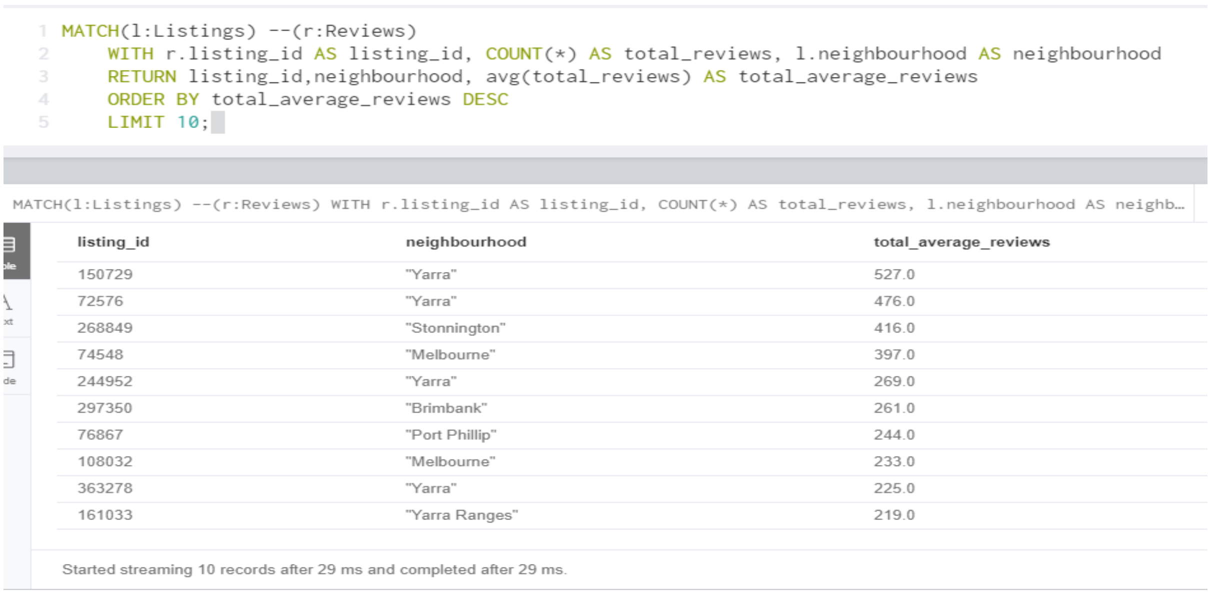


```
MATCH (r:Reviews) <- [:gaveReviews]-(l:Listings)
    WHERE date(r.date) >= date("2017-01-01") and date(r.date) < date("2018-01-01")
    RETURN COUNT(l.name);
```

```
MATCH (r:Reviews) <- [:gaveReviews]-(l:Listings) WHERE date(r.date) >= date("2017-01-01") and date(r.date) < date("20…
```

COUNT(l.name)
1233

12) What are the top 10 most popular neighbourhoods based on the total average reviews?



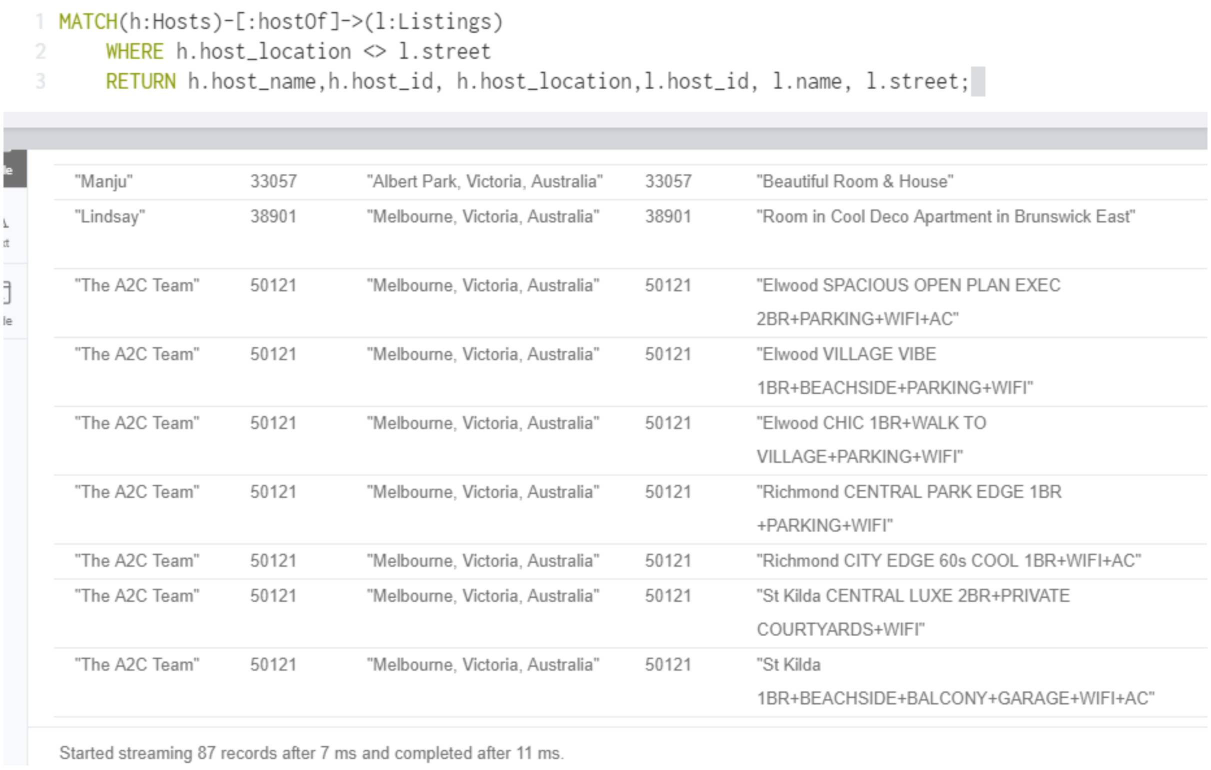
```
MATCH(l:Listings) --(r:Reviews)
    WITH r.listing_id AS listing_id, COUNT(*) AS total_reviews, l.neighbourhood AS neighbourhood
    RETURN listing_id,neighbourhood, avg(total_reviews) AS total_average_reviews
    ORDER BY total_average_reviews DESC
    LIMIT 10;
```

```
MATCH(l:Listings) --(r:Reviews) WITH r.listing_id AS listing_id, COUNT(*) AS total_reviews, l.neighbourhood AS neighb…
```

listing_id	neighbourhood	total_average_reviews
150729	"Yarra"	527.0
72576	"Yarra"	476.0
268849	"Stonnington"	416.0
74548	"Melbourne"	397.0
244952	"Yarra"	269.0
297350	"Brimbank"	261.0
76867	"Port Phillip"	244.0
108032	"Melbourne"	233.0
363278	"Yarra"	225.0
161033	"Yarra Ranges"	219.0

Started streaming 10 records after 29 ms and completed after 29 ms.

13) Find hosts whose location are different from their listings. Show the host name, host location, listing name, and listing location.

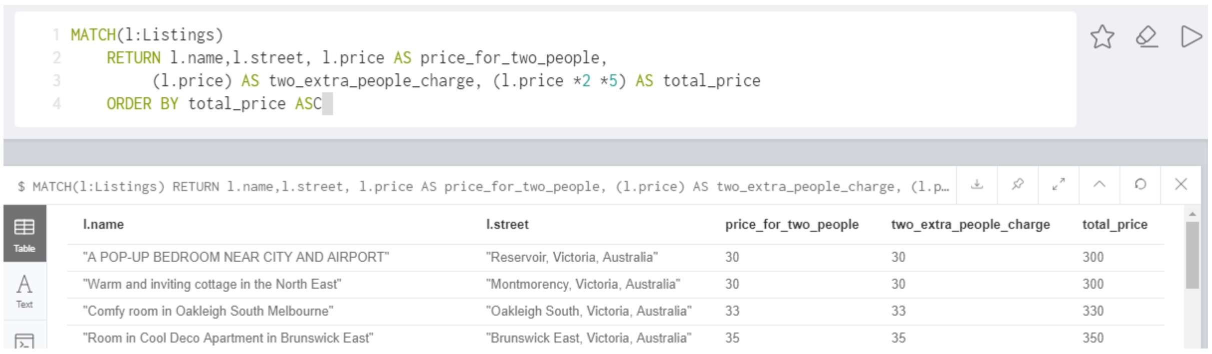


```
MATCH(h:Hosts)-[:hostOf]->(l:Listings)
    WHERE h.host_location <> l.street
    RETURN h.host_name,h.host_id, h.host_location,l.host_id, l.name, l.street;
```

"Manju"	33057	"Albert Park, Victoria, Australia"	33057	"Beautiful Room & House"
"Lindsay"	38901	"Melbourne, Victoria, Australia"	38901	"Room in Cool Deco Apartment in Brunswick East"
"The A2C Team"	50121	"Melbourne, Victoria, Australia"	50121	"Elwood SPACIOUS OPEN PLAN EXEC 2BR+PARKING+WIFI+AC"
"The A2C Team"	50121	"Melbourne, Victoria, Australia"	50121	"Elwood VILLAGE VIBE 1BR+BEACHSIDE+PARKING+WIFI"
"The A2C Team"	50121	"Melbourne, Victoria, Australia"	50121	"Elwood CHIC 1BR+WALK TO VILLAGE+PARKING+WIFI"
"The A2C Team"	50121	"Melbourne, Victoria, Australia"	50121	"Richmond CENTRAL PARK EDGE 1BR +PARKING+WIFI"
"The A2C Team"	50121	"Melbourne, Victoria, Australia"	50121	"Richmond CITY EDGE 60s COOL 1BR+WIFI+AC"
"The A2C Team"	50121	"Melbourne, Victoria, Australia"	50121	"St Kilda CENTRAL LUXE 2BR+PRIVATE COURTYARDS+WIFI"
"The A2C Team"	50121	"Melbourne, Victoria, Australia"	50121	"St Kilda 1BR+BEACHSIDE+BALCONY+GARAGE+WIFI+AC"

Started streaming 87 records after 7 ms and completed after 11 ms.

14) Assuming that each accommodation only accepts two guests, calculate the price of each accommodation for four people staying for five nights. Display only the accommodation name, location, price per night, extra people charge, and total price. Rank the accommodation from the cheapest price.

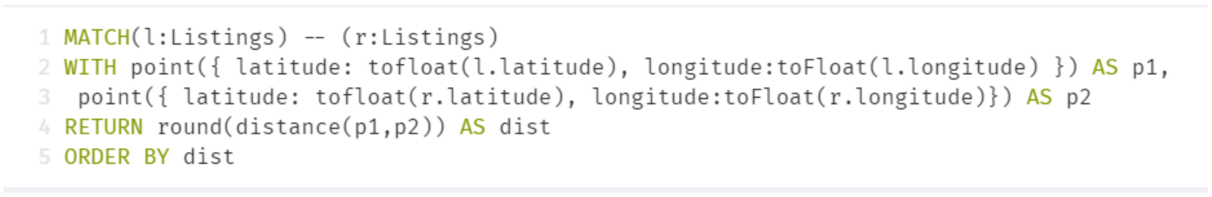


```
MATCH(l:Listings)
   RETURN l.name,l.street, l.price AS price_for_two_people,
        (l.price) AS two_extra_people_charge, (l.price *2 *5) AS total_price
   ORDER BY total_price ASC
```

```
$ MATCH(l:Listings) RETURN l.name,l.street, l.price AS price_for_two_people, (l.price) AS two_extra_people_charge, (l.p…
```

l.name	l.street	price_for_two_people	two_extra_people_charge	total_price
"A POP-UP BEDROOM NEAR CITY AND AIRPORT"	"Reservoir, Victoria, Australia"	30	30	300
"Warm and inviting cottage in the North East"	"Montmorency, Victoria, Australia"	30	30	300
"Comfy room in Oakleigh South Melbourne"	"Oakleigh South, Victoria, Australia"	33	33	330
"Room in Cool Deco Apartment in Brunswick East"	"Brunswick East, Victoria, Australia"	35	35	350

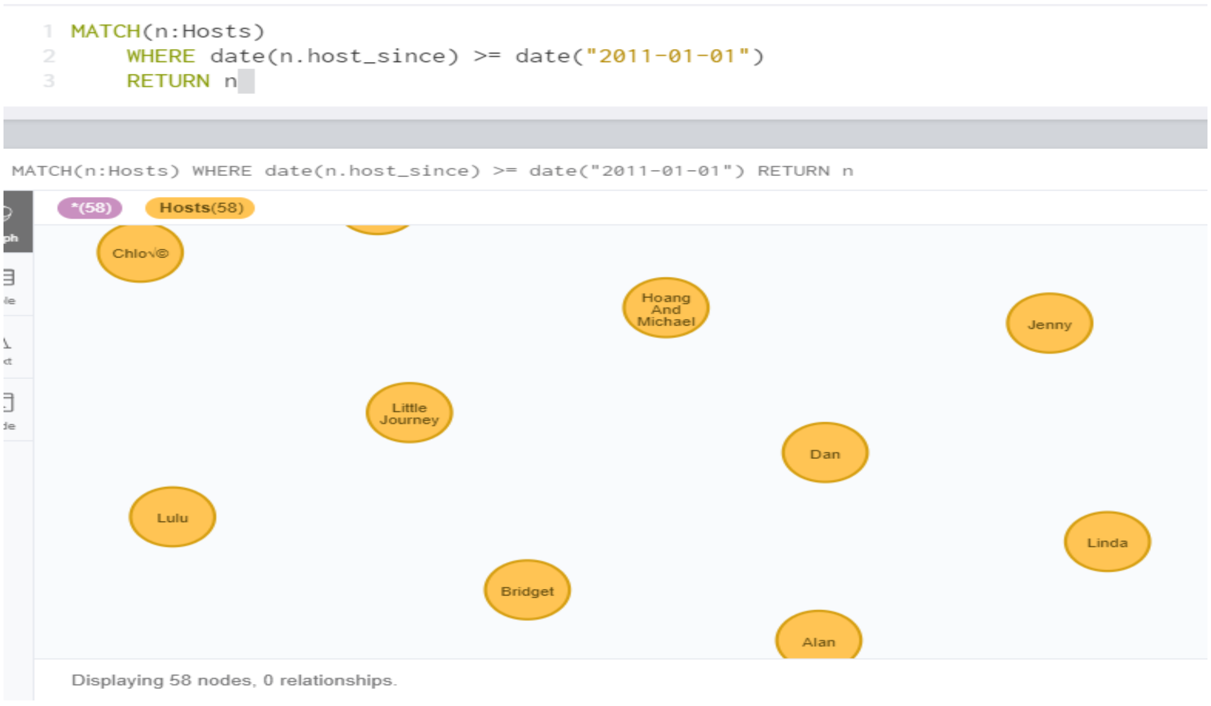
15) For each listing, rank other listings that are close to each other by their locations. You will need to use the longitude and latitude to calculate the distance between listings.



```
MATCH(l:Listings) -- (r:Listings)
WITH point({ latitude: tofloat(l.latitude), longitude:toFloat(l.longitude) }) AS p1,
 point({ latitude: tofloat(r.latitude), longitude:toFloat(r.longitude)}) AS p2
RETURN round(distance(p1,p2)) AS dist
ORDER BY dist
```

<u>Additional Queries:</u>

1) List host who registered after 2010 ?

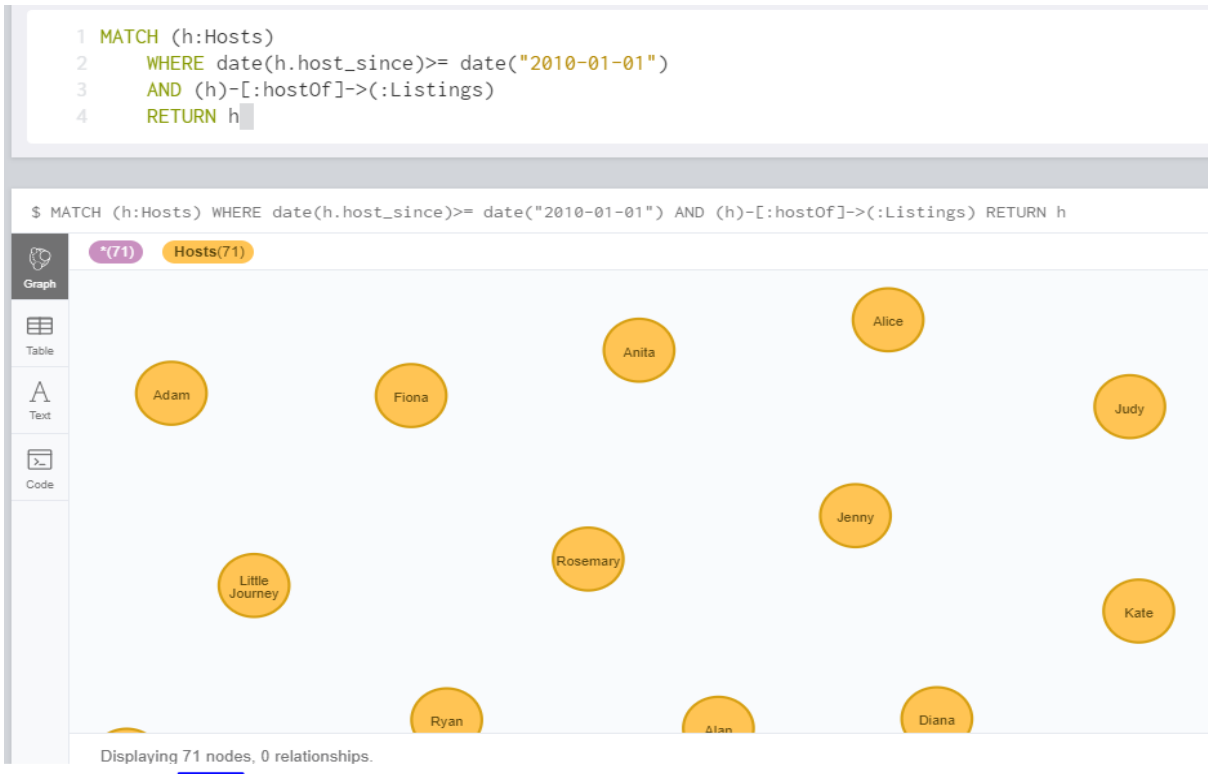


```
MATCH(n:Hosts)
    WHERE date(n.host_since) >= date("2011-01-01")
    RETURN n
```

```
MATCH(n:Hosts) WHERE date(n.host_since) >= date("2011-01-01") RETURN n
```

Displaying 58 nodes, 0 relationships.

2) How many hosts have listings since 2010?

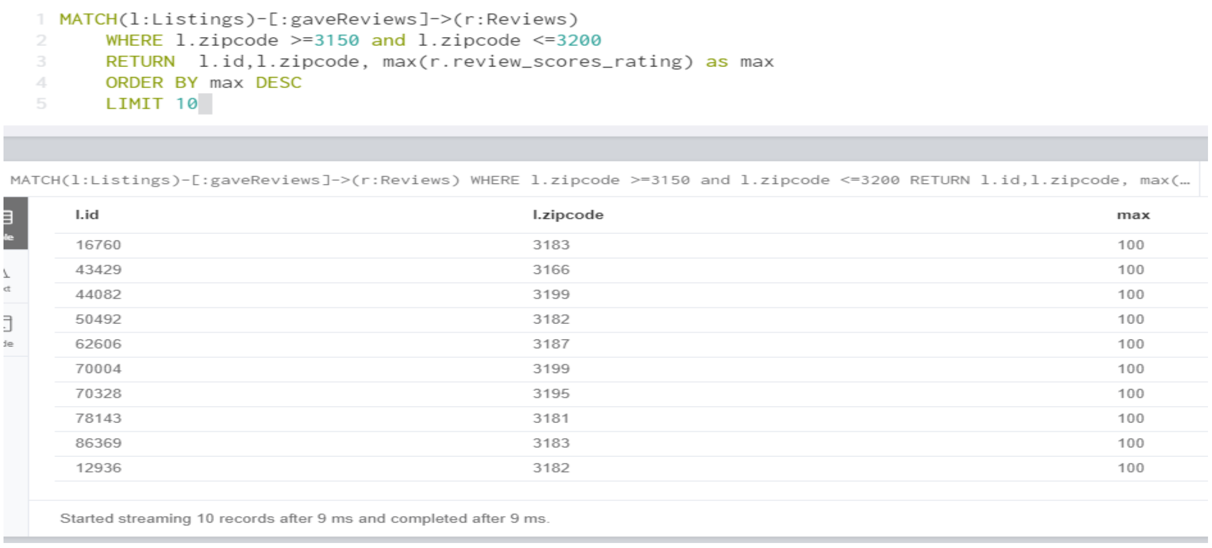


```
MATCH (h:Hosts)
    WHERE date(h.host_since)>= date("2010-01-01")
    AND (h)-[:hostOf]->(:Listings)
    RETURN h
```

```
$ MATCH (h:Hosts) WHERE date(h.host_since)>= date("2010-01-01") AND (h)-[:hostOf]->(:Listings) RETURN h
```

Displaying 71 nodes, 0 relationships.

3) Display top 5 listings between the zip code 3150 and 3200 which has highest review rating?



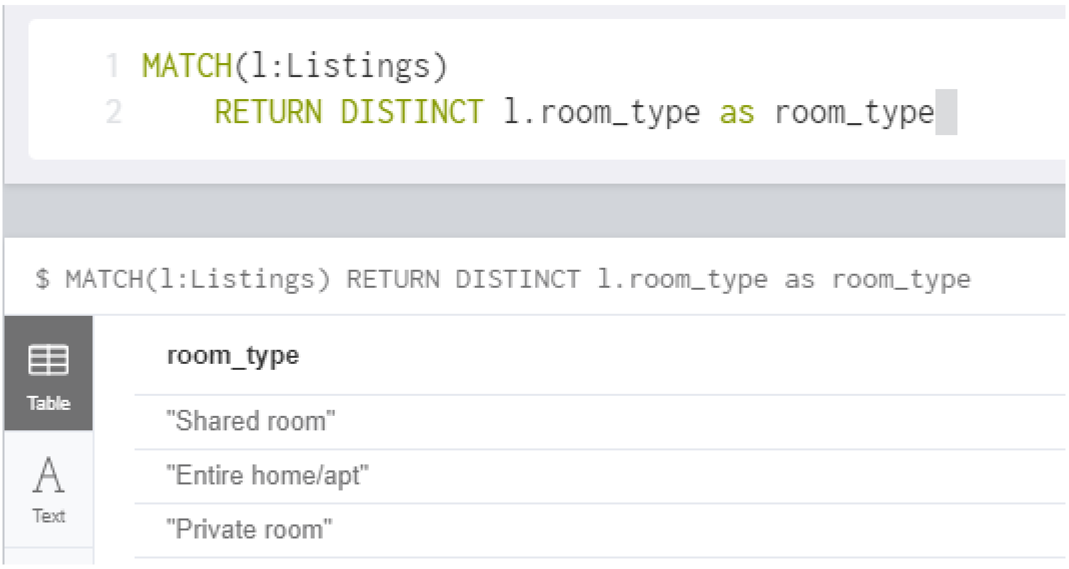
```
MATCH(l:Listings)-[:gaveReviews]->(r:Reviews)
    WHERE l.zipcode >=3150 and l.zipcode <=3200
    RETURN  l.id,l.zipcode, max(r.review_scores_rating) as max
    ORDER BY max DESC
    LIMIT 10
```

```
MATCH(l:Listings)-[:gaveReviews]->(r:Reviews) WHERE l.zipcode >=3150 and l.zipcode <=3200 RETURN l.id,l.zipcode, max(…
```

l.id	l.zipcode	max
16760	3183	100
43429	3166	100
44082	3199	100
50492	3182	100
62606	3187	100
70004	3199	100
70328	3195	100
78143	3181	100
86369	3183	100
12936	3182	100

Started streaming 10 records after 9 ms and completed after 9 ms.

4) List unique room types in listings?

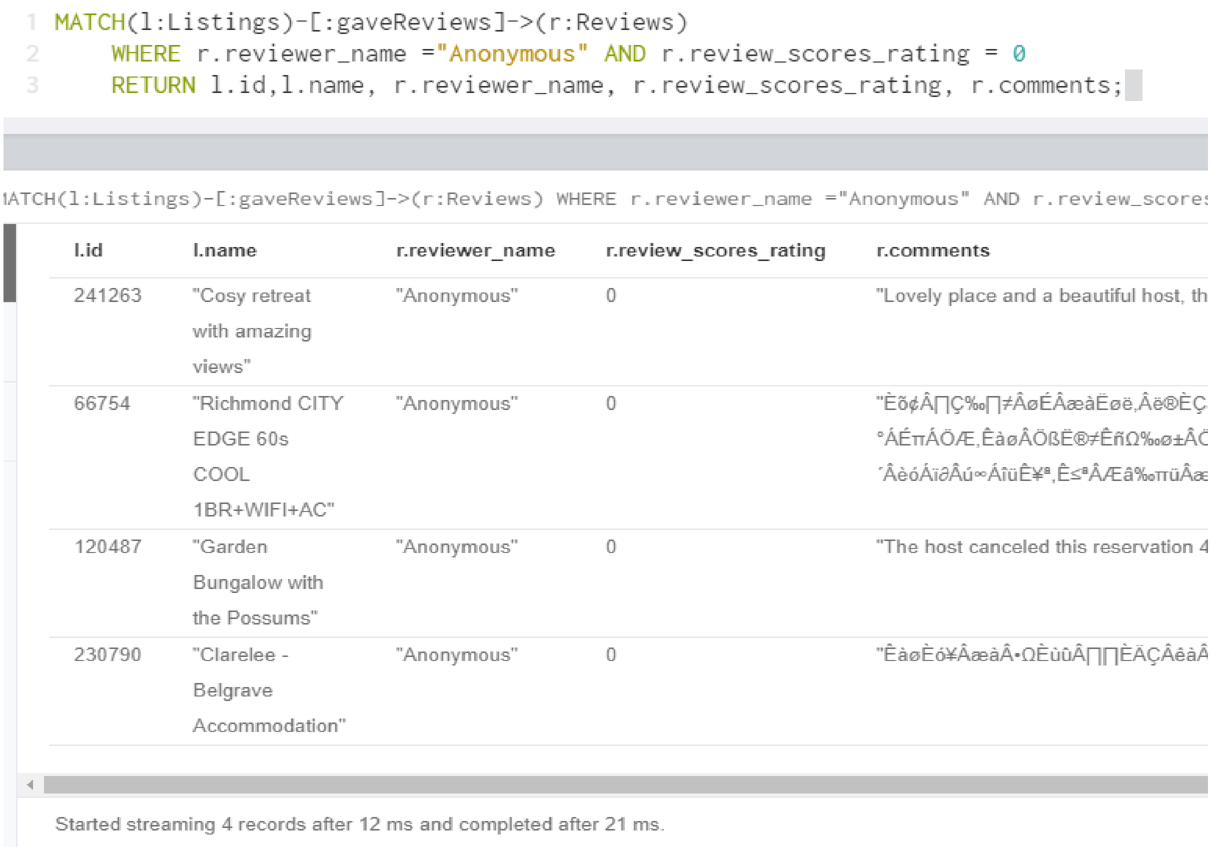


```
MATCH(l:Listings)
    RETURN DISTINCT l.room_type as room_type
```

```
$ MATCH(l:Listings) RETURN DISTINCT l.room_type as room_type
```

room_type
"Shared room"
"Entire home/apt"
"Private room"

5) List listings which has bad review rating given by anonymous reviewers?



```
MATCH(l:Listings)-[:gaveReviews]->(r:Reviews)
    WHERE r.reviewer_name ="Anonymous" AND r.review_scores_rating = 0
    RETURN l.id,l.name, r.reviewer_name, r.review_scores_rating, r.comments;
```

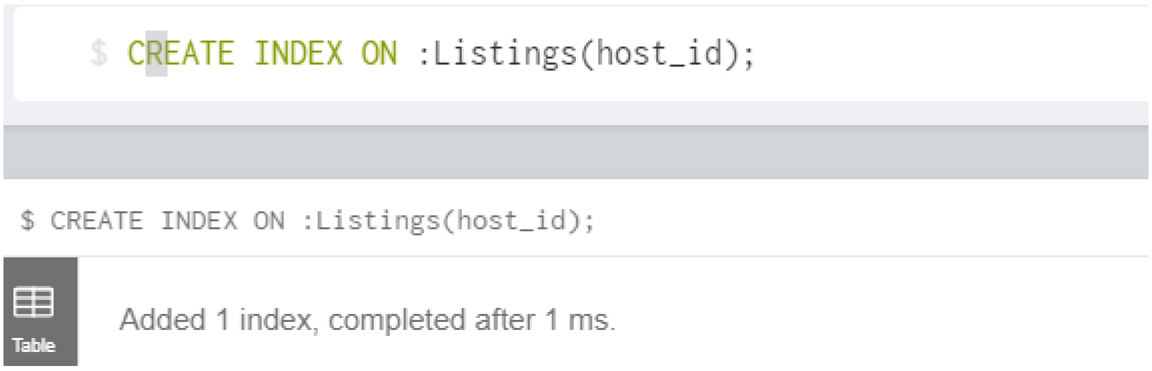
```
IATCH(l:Listings)-[:gaveReviews]->(r:Reviews) WHERE r.reviewer_name ="Anonymous" AND r.review_score
```

l.id	l.name	r.reviewer_name	r.review_scores_rating	r.comments
241263	"Cosy retreat with amazing views"	"Anonymous"	0	"Lovely place and a beautiful host, th
66754	"Richmond CITY EDGE 60s COOL 1BR+WIFI+AC"	"Anonymous"	0	"Èõ¢Â∏Ç‰∏≠ÂøÉÂæàËøë,Âë®ÈÇ °ÁÉπÁÖÆ,ÊàøÂÖßË®≠ÊñΩ‰ø±ÂÖ ˝ÂèóÁïÇÂú∞ÁîüÊ¥ª,Ê≤ªÂÆâ‰πüÂæ
120487	"Garden Bungalow with the Possums"	"Anonymous"	0	"The host canceled this reservation 4
230790	"Clarelee - Belgrave Accommodation"	"Anonymous"	0	"ÊàøÈó¥ÂæàÂ•ΩÈùûÂ∏∏ÈÄÇÂêàÂ

Started streaming 4 records after 12 ms and completed after 21 ms.

CREATE INDEXES:

1)



```
$ CREATE INDEX ON :Listings(host_id);
```

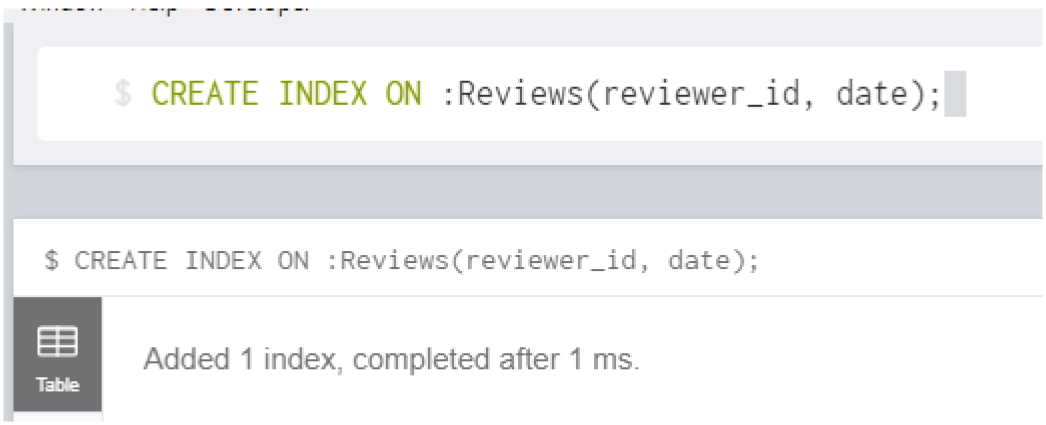
```
$ CREATE INDEX ON :Listings(host_id);
```

Table

```
Added 1 index, completed after 1 ms.
```

Here, simple index is created on Listings labels using the "host-id" properties. This property "host_id" is selected based on its usage in domain-based queries in TaskC2. In Query 3 & 14, the host_id property in Listings node is used to retrieve the values. Therefore, creating index for "host_id" property improves the query efficiency and optimization.

2)



```
$ CREATE INDEX ON :Reviews(reviewer_id, date);
```

```
$ CREATE INDEX ON :Reviews(reviewer_id, date);
```

Table

```
Added 1 index, completed after 1 ms.
```

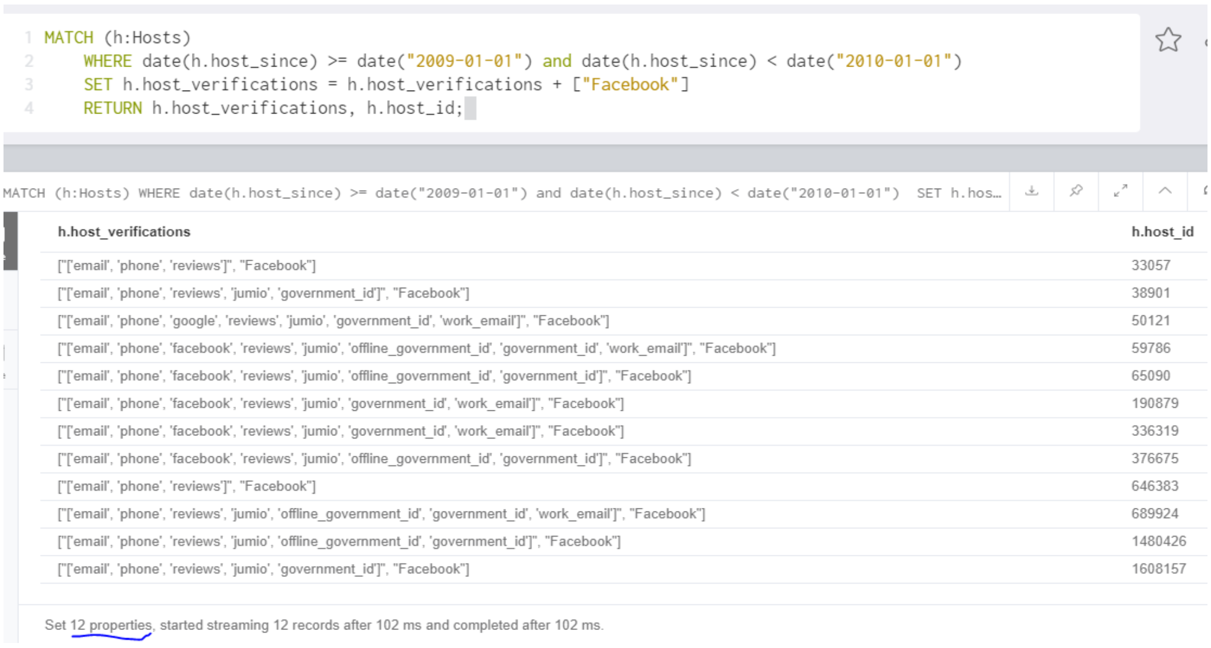
Here composite index is created on Reviews label using "reviewer_id" and "date" label. These properties are chosen based on the domain-based queries in Task C2 and C3. In task C2 for Query 3,5 & 11, in task C3 for Query 4 the data are retrieved either using reviewer-id or date or both. Therefore, creating composite index for "reviewer_id, date" properties improves the query efficiency and optimization

TASK C3:

1) Go to AirBnB website and add three new listings, including the hosts details and some related reviews of the listings you chose. The IDs in this case can be assigned manually by yourself.

   Added 3 new listings and respective hosts and reviews in the database.

2) Update the host verification for those who registered in 2009 and add Facebook to the list of existing verifications.



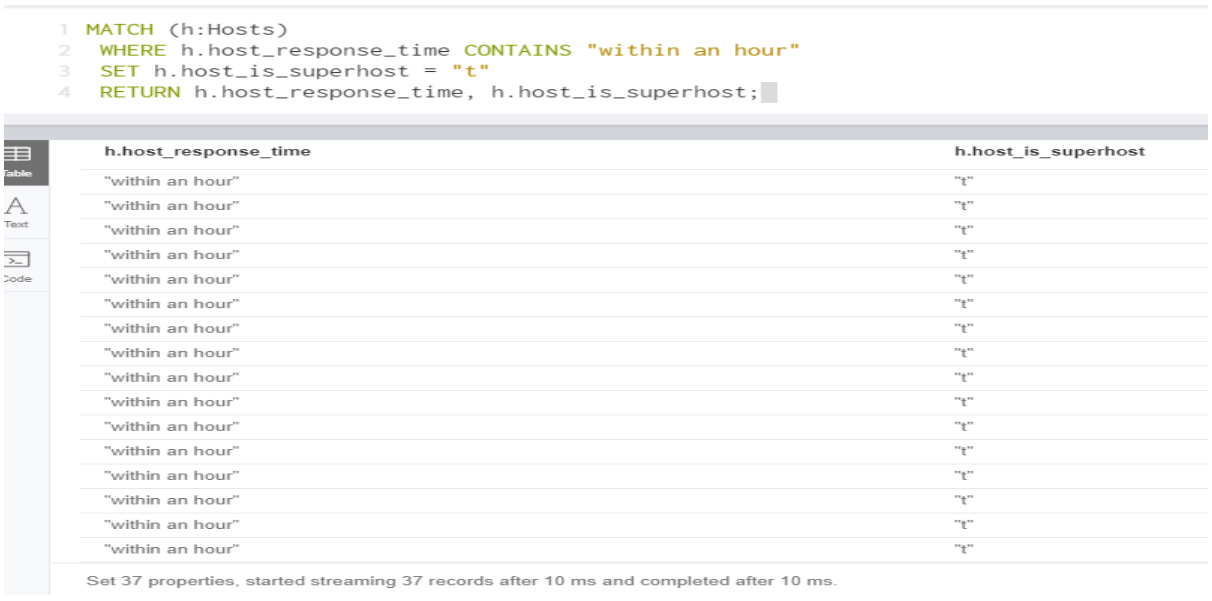
```
MATCH (h:Hosts)
    WHERE date(h.host_since) >= date("2009-01-01") and date(h.host_since) < date("2010-01-01")
    SET h.host_verifications = h.host_verifications + ["Facebook"]
    RETURN h.host_verifications, h.host_id;
```

```
MATCH (h:Hosts) WHERE date(h.host_since) >= date("2009-01-01") and date(h.host_since) < date("2010-01-01")  SET h.hos…
```

h.host_verifications	h.host_id
["['email', 'phone', 'reviews']", "Facebook"]	33057
["['email', 'phone', 'reviews', 'jumio', 'government_id']", "Facebook"]	38901
["['email', 'phone', 'google', 'reviews', 'jumio', 'government_id', 'work_email']", "Facebook"]	50121
["['email', 'phone', 'facebook', 'reviews', 'jumio', 'offline_government_id', 'government_id', 'work_email']", "Facebook"]	59786
["['email', 'phone', 'facebook', 'reviews', 'jumio', 'offline_government_id', 'government_id']", "Facebook"]	65090
["['email', 'phone', 'facebook', 'reviews', 'jumio', 'government_id', 'work_email']", "Facebook"]	190879
["['email', 'phone', 'facebook', 'reviews', 'jumio', 'government_id', 'work_email']", "Facebook"]	336319
["['email', 'phone', 'facebook', 'reviews', 'jumio', 'offline_government_id', 'government_id']", "Facebook"]	376675
["['email', 'phone', 'reviews']", "Facebook"]	646383
["['email', 'phone', 'reviews', 'jumio', 'offline_government_id', 'government_id', 'work_email']", "Facebook"]	689924
["['email', 'phone', 'reviews', 'jumio', 'offline_government_id', 'government_id']", "Facebook"]	1480426
["['email', 'phone', 'reviews', 'jumio', 'government_id']", "Facebook"]	1608157

Set 12 properties, started streaming 12 records after 102 ms and completed after 102 ms.

3) Update hosts who respond “within an hour” to a super host. For this update you may only use the “host response time” and “host is a super host” information.

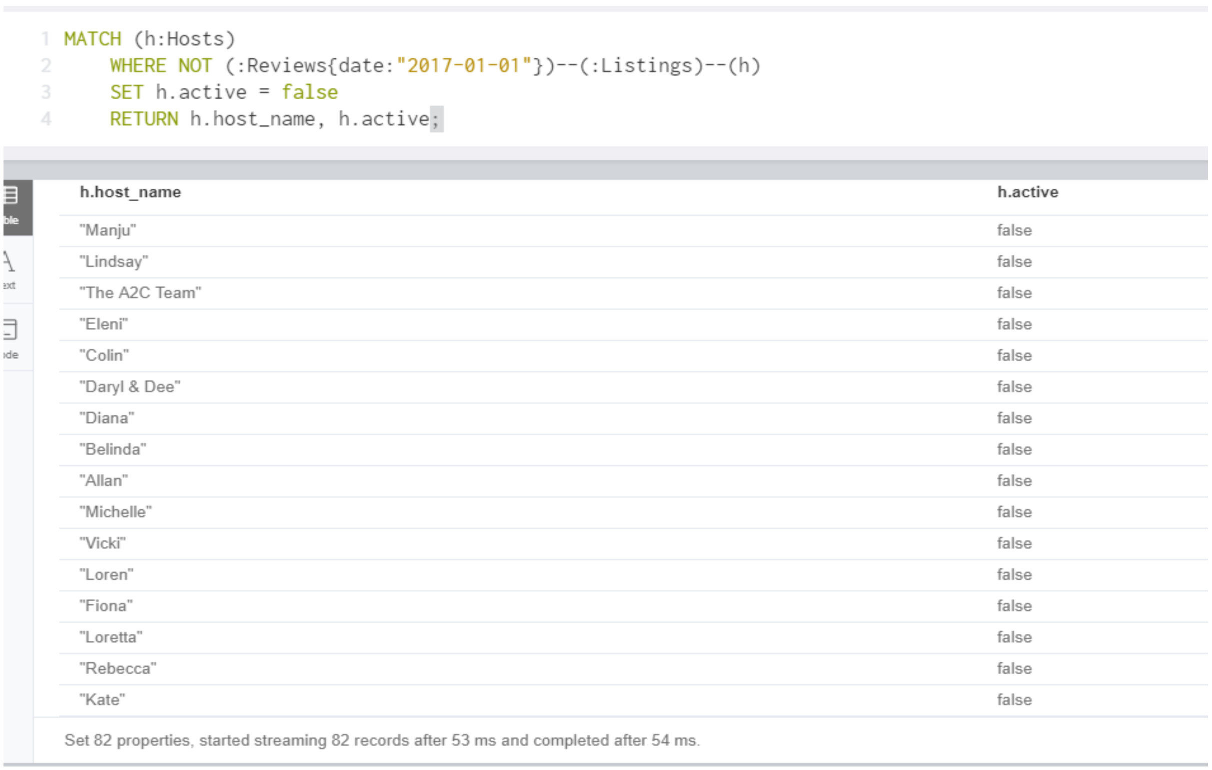


```
MATCH (h:Hosts)
 WHERE h.host_response_time CONTAINS "within an hour"
 SET h.host_is_superhost = "t"
 RETURN h.host_response_time, h.host_is_superhost;
```

h.host_response_time	h.host_is_superhost
"within an hour"	"t"
"within an hour"	"t"
"within an hour"	"t"
"within an hour"	"t"
"within an hour"	"t"
"within an hour"	"t"
"within an hour"	"t"
"within an hour"	"t"
"within an hour"	"t"
"within an hour"	"t"
"within an hour"	"t"
"within an hour"	"t"
"within an hour"	"t"
"within an hour"	"t"
"within an hour"	"t"
"within an hour"	"t"

Set 37 properties, started streaming 37 records after 10 ms and completed after 10 ms.

4) Update hosts who do not receive any reviews for their accommodation since 2017 and add a new property called active. This new property accepts Boolean value.

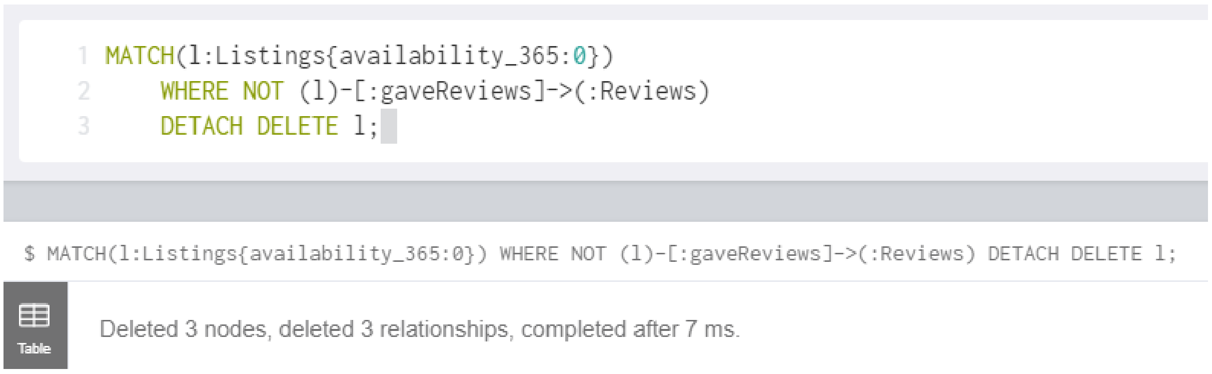


```
MATCH (h:Hosts)
    WHERE NOT (:Reviews{date:"2017-01-01"})--(:Listings)--(h)
    SET h.active = false
    RETURN h.host_name, h.active;
```

h.host_name	h.active
"Manju"	false
"Lindsay"	false
"The A2C Team"	false
"Eleni"	false
"Colin"	false
"Daryl & Dee"	false
"Diana"	false
"Belinda"	false
"Allan"	false
"Michelle"	false
"Vicki"	false
"Loren"	false
"Fiona"	false
"Loretta"	false
"Rebecca"	false
"Kate"	false

Set 82 properties, started streaming 82 records after 53 ms and completed after 54 ms.

5) Delete all listings with zero availability and have no reviews.



```
MATCH(l:Listings{availability_365:0})
    WHERE NOT (l)-[:gaveReviews]->(:Reviews)
    DETACH DELETE l;
```

```
$ MATCH(l:Listings{availability_365:0}) WHERE NOT (l)-[:gaveReviews]->(:Reviews) DETACH DELETE l;
```

```
Deleted 3 nodes, deleted 3 relationships, completed after 7 ms.
```

Task C4

This task is about explaining briefly about how Neo4j graph database is used widely in real world with various terms and features. It is popularly used in various domains like

- Used by Airbnb and Expedia for accommodation booking system
- Used by Walmart for serving real time recommendations on their websites in 11 countries and helps to understand online shopper's behaviour and relationships between customers and products.
- Used by eBay, a multinational e-commerce corporation to help merchants by matching orders with same-day delivery couriers quickly.
- Popular movie recommendation site.

In addition to that, Neo4j is popularly used in Social networking, geospatial intelligence and searching algorithms. But here, report also explains how Neo4j used in AirBnb.

Workflow of Airbnb using Neo4j database:

Airbnb generates and manages multiple data points to provide hospitality service for the people to lease or rent lodging. The company uses "Airbnb's Hive Data Warehouse" that accumulates large amount of data with more than 200,000 tables across multiple clusters.

Airflow methodology is a platform that programmatically schedule and monitor data pipelines using directed acyclic graphs (DAG's) of tasks. Using this methodology, the data is pushed to the python eventually it is pushed to the neo4j by Neo4j driver. In order to provide live update to the customer about current trending lodges that has better pricing, the data is pushed from Hive to the neo4j every day to get live updates in the neo4j database and that updates are showed to the customer.

Airbnb focus mainly on showing data based on customer expectation at the time of using the application. Each user expects different kind of lodging at different location, in order attain this, they selected Neo4j as it captures the "relationship relevancy" between the data resources and people. Here the relationships are extremely important to know who consumes and created the resources and moreover the relevancy of relationship provides equal standing between the nodes and relationships.

It is also important that based on the customer relevant search the data must be discovered. This is possible only when proper decision making happens in the large amount of data in the live database. This can be done using "Data Portal", a tool that help with data discovery and decision making in Airbnb. Data portal consume data entered in the search entry and convert it into meaningful search way and sends it to the relevance search indices to retrieve data resources.

Using the relevant data sources, the graph built with nodes and relationships and forms the data and shows it to the customer based on relevant search. Since the database updates with the live data, there is the possibility to occur graph flickering, this is also due to the certain types of relationships. To avoid graph flickering, Neo4j classified relationships into two "persistent" and "transient". The data the expires in certain time period are in transient relationship that make graph flicker and it is was resolved by expanding the time period.

In this way Airbnb works on providing accommodation and hospitality services to the customer. Similarly, Expedia, uses large amount of data warehouse and push the data into the Neo4j for live updates to the customer.

Neo4j also makes the task easier for database designers. Some of the advantages for developers to prefer Neo4j database are,

- It is a NoSQL database, so no structure needs to be maintained
- Can design base on schema less approach
- No need to rely on join operations and foreign keys to handle data

MonashBnb can use graph database, as it is easy to retrieve the expected data and able to handle the level of control if the database is well designed. This Neo4j has some main key product features that plays major role in various industries that uses Neo4j, MonashBnb can follow those features to improve their graph database, they are

- Ensures data consistency and performance as Neo4j has native graph storage
- With real time data set, ensure MonashBnb graph database can perform millions of hops per second as it has the native graph processing
- Ensure it provides high scalability and availability as graph contains optimized vertical and horizontal scaling.

Neo4j performs better integration by providing better drivers and API's for popular language like Java and also deals with powerful and expressive query language because when compare to other structured query language, Neo4j requires very less code to retrieve large amount of data from the datasets with high data integrity and ACID transactions. In addition to that, as it provides built in ETL, it will be easy to import various plugins from other databases.

References

Real-Time Recommendations in the Real World - Neo4j Graph Database Platform. (2019). Retrieved 22 October 2019, from https://neo4j.com/business-edge/real-time-recommendations-in-the-real-world/

Airbnb Democratizes Data Discovery with the Neo4j Graph Platform. (2019). Retrieved 22 October 2019, from https://neo4j.com/blog/democratizing-data-discovery-airbnb/

(2019). Retrieved 22 October 2019, from https://www.quora.com/What-are-real-world-problems-solved-with-a-graph-database

Graph Database Real-World Examples ⋆ Bitnine Global Inc. (2019). Retrieved 22 October 2019, from https://bitnine.net/blog-graph-database/graph-database-real-world-examples/

What Is a Graph Database and Property Graph | Neo4j. (2019). Retrieved 22 October 2019, from https://neo4j.com/developer/graph-database/

Neo4J - Features & Advantages - Tutorialspoint. (2019). Retrieved 22 October 2019, from https://www.tutorialspoint.com/neo4j/neo4j_features_advantages.htm