FIT5137: Advanced Database Technology

Assignment 2

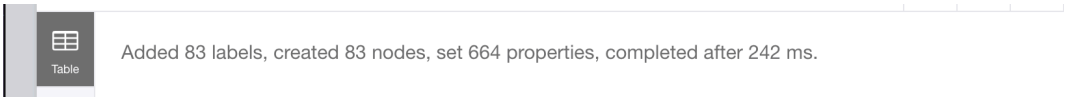
Neo4j

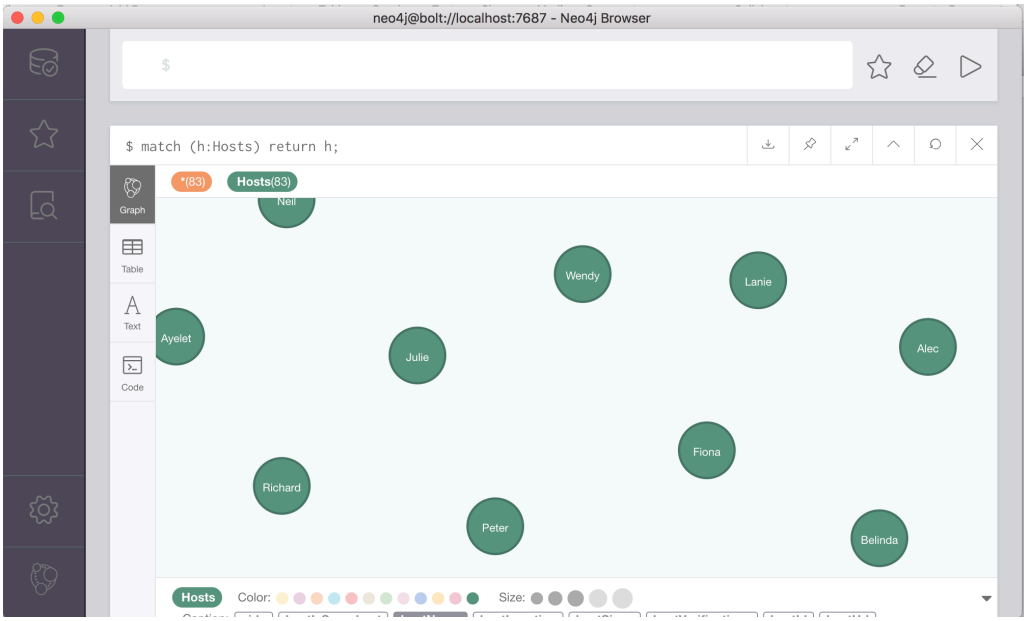
Sarah Alshaikhmubarak - 29462924

C.1. Database Design.

Import Host:

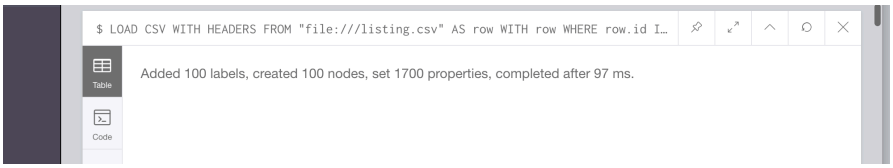
```
LOAD CSV WITH HEADERS FROM "file:///host.csv"
AS row
WITH row WHERE row.host_id IS NOT NULL
MERGE (h:Hosts {hostId: row.host_id})
ON CREATE SET h.hostUrl = row.host_url, h.hostName = row.host_name, h.hostVerifications =
apoc.convert.fromJsonList(row.host_verifications), h.hostSince = apoc.date.parse(row.host_since, 'ms','yyyy-
MM-dd'), h.hostLocation = row.host_location, h.hostResponseTime = row.host_response_time,
h.hostIsSuperhost =(case row.host_is_superhost when 't' then true else false end);
```

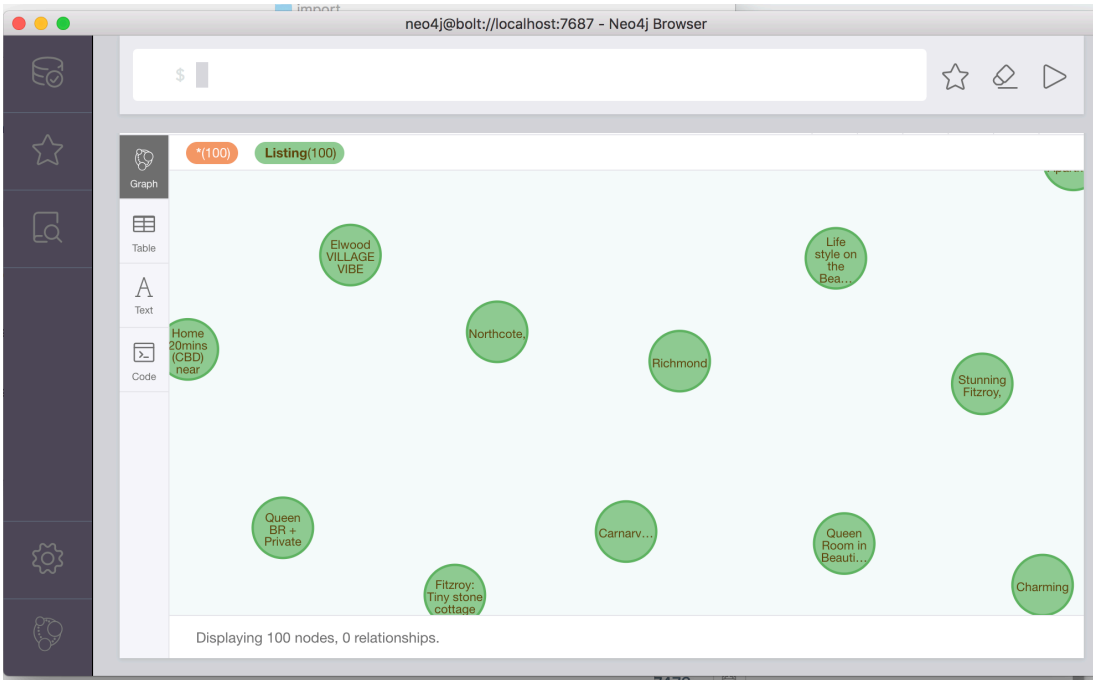




Import Listing:

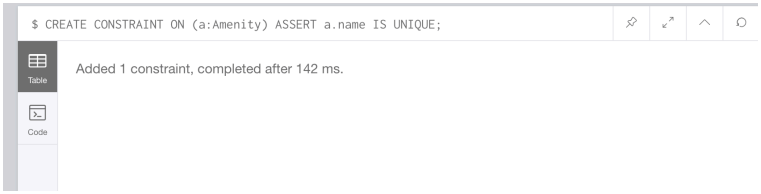
```
LOAD CSV WITH HEADERS FROM "file:///listing.csv"
AS row
WITH row WHERE row.id IS NOT NULL
MERGE (l:Listing {listingId: row.id})
ON CREATE SET l.name = row.name, l.summary = row.summary, l.listingUrl = row.listing_url, l.pictureUrl
= row.picture_url, l.hostId = row.host_id, l.neighbourhood = row.neighbourhood, l.street = row.street,
l.zipcode = toInteger(row.zipcode), l.latitude = toFloat(row.latitude), l.longitude =toFloat(row.longitude),
l.roomType = row.room_type, l.price= toFloat(row.price), l.extraPeople =
toFloat(substring(row.extra_people,1)), l.minimumNights = toInteger(row.minimum_nights),
l.calculatedHostListingsCount = toInteger(row.calculated_host_listings_count), l.availability365 =
toInteger(row.availability_365);
```





Create Constraint ON Amenity:

CREATE CONSTRAINT ON (a:Amenity) ASSERT a.name IS UNIQUE;

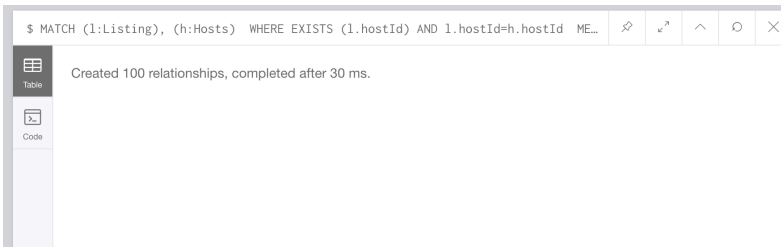


Import Amenities:

```
LOAD CSV WITH HEADERS FROM "file:///listing.csv" AS row
WITH row WHERE row.id IS NOT NULL
MATCH (l:Listing {listingId: row.id})
WITH l, split(replace(replace(replace(row.amenities, "{", ""), "}", ""), "\"", ""), ",") AS amenities
UNWIND amenities AS amenity
MERGE (a:Amenity {name: amenity})
MERGE (l)-[:HAS]->(a)
```

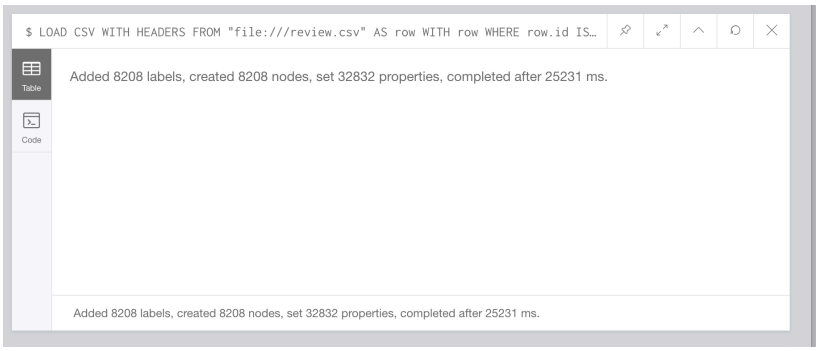
Create a relationship between host and listing:

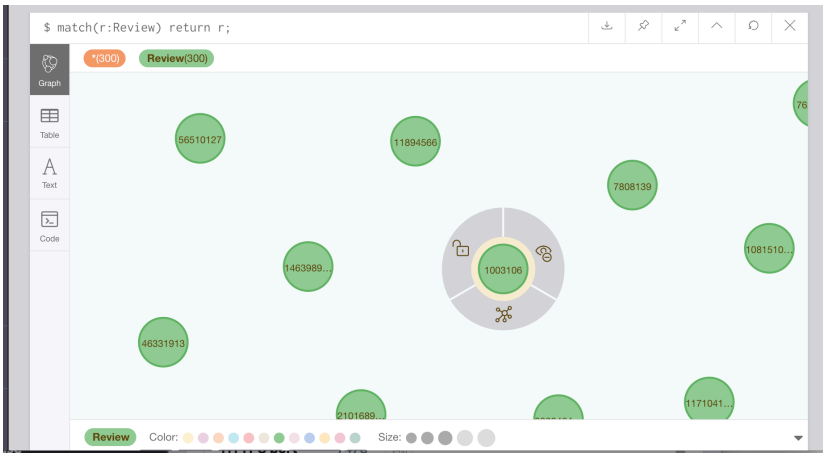
```
MATCH (l:Listing), (h:Hosts)
WHERE EXISTS (l.hostId) AND l.hostId=h.hostId
MERGE (h)-[:HOSTS]->(l);
```



Import Review:

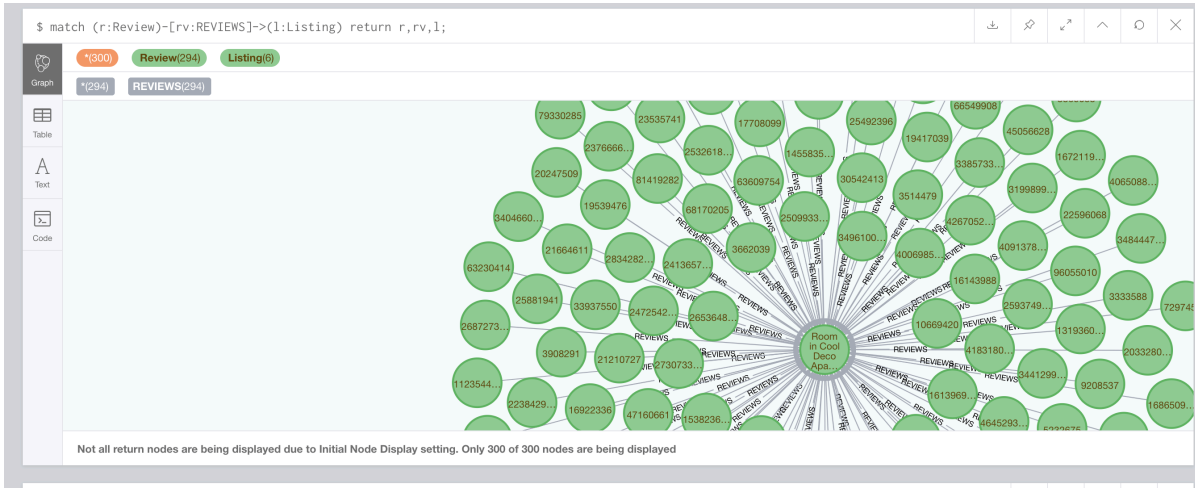
```
LOAD CSV WITH HEADERS FROM "file:///review.csv" AS row
WITH row WHERE row.id IS NOT NULL
MERGE (r:Review {id: row.id})
ON CREATE SET r.date =apoc.date.parse(row.date,'ms','yyyy-MM-dd'),
    r.comments = row.comments, r.listingId = row.listing_id;
```





Relationship between review and listing:

LOAD CSV WITH HEADERS FROM "file:///review.csv" AS
row
MATCH (r:Review {id: row.id})
MATCH (l:Listing {listingId: row.listing_id})
CREATE (r)-[:REVIEWS{reviewScoresRating:toInteger(row.review_scores_rating)}]->(l);

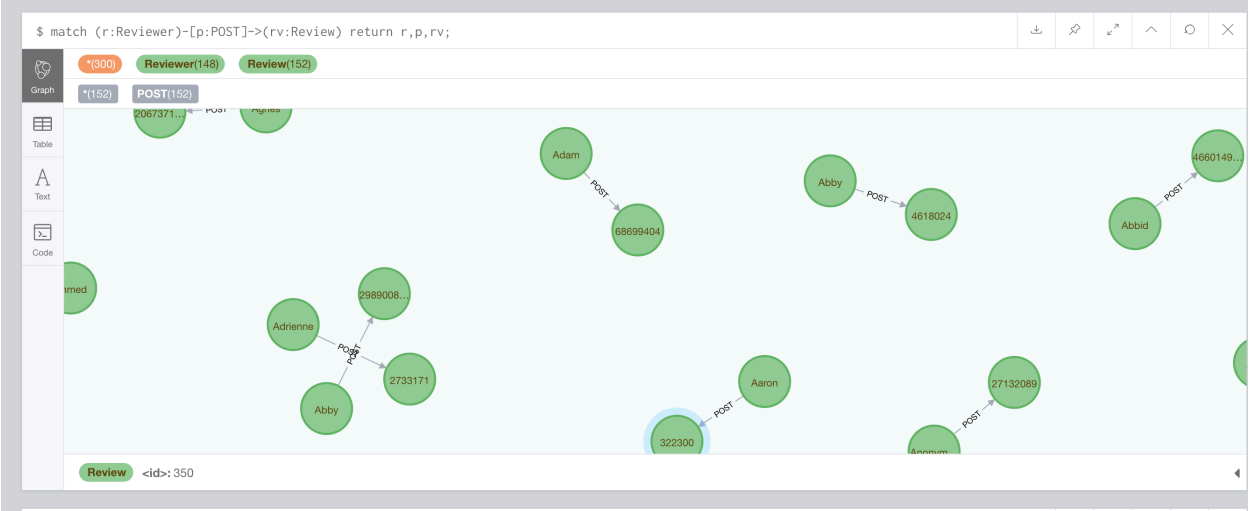


Create Constraint:

CREATE CONSTRAINT ON (rv:Reviewer) ASSERT rv.reviewerId IS UNIQUE;

Import Reviewers:

LOAD CSV WITH HEADERS FROM "file:///review.csv" AS row
WITH row WHERE row.id IS NOT NULL
MATCH (r:Review {id: row.id})
MERGE (rv:Reviewer {reviewerId:row.reviewer_id})
SET rv.reviewerName = row.reviewer_name
MERGE (rv)-[:POST]->(r)

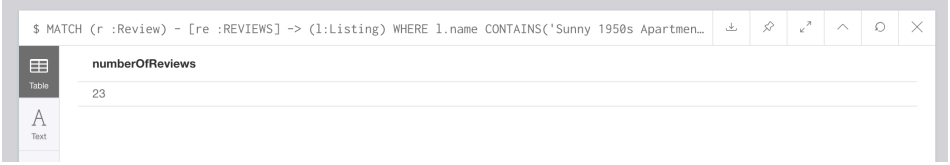


To develop MonashBnB database using neo4j, I have created five nodes, which are: Hosts, Listing, Review, Reviewer, and Amenity. The reason for designing these five is that each host should be a node contain each host's information and the same for Listing. For Amenity, I have separated it from Listing, and the reason is that the Amenity type is an object and I think it's better when querying. And for Reviewer, because the relationship between review and reviewer is one two many so it is better to separate them. For relationships, four relationships have been created to facilitate querying. For date and array type, I have used APOC plugin to do that. Thus, it's better to install APOC in the neo4j project.

C.2. Queries.

1. How many reviews does "Sunny 1950s Apartment, St Kilda East" have?

MATCH (r :Review) – [re :REVIEWS] -> (l:Listing) WHERE l.name CONTAINS('Sunny 1950s Apartment, St Kilda East') RETURN count(r) As numberOfReviews;

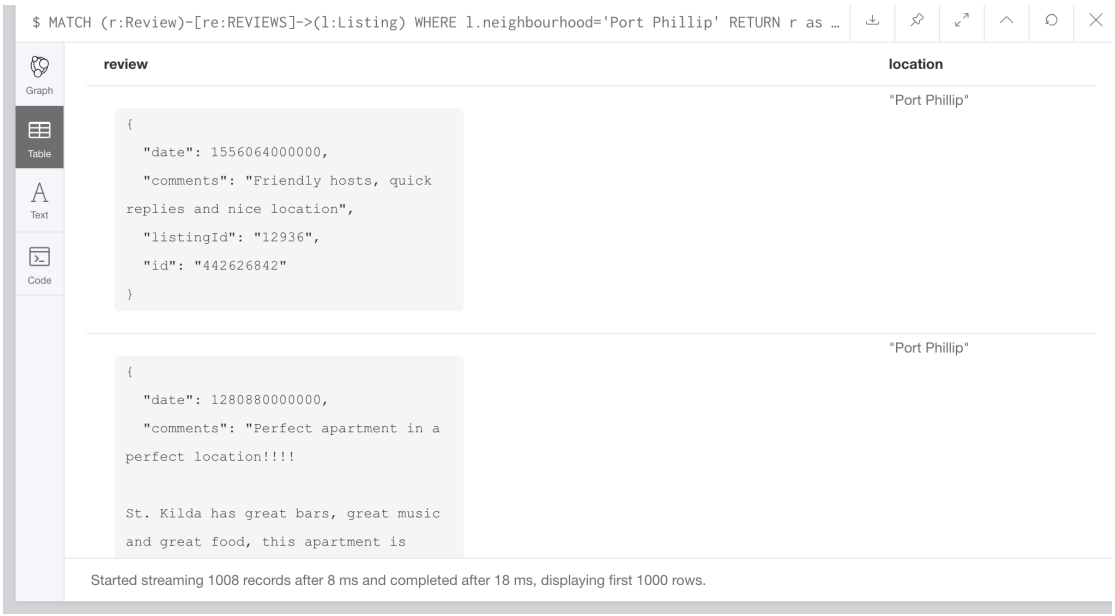


Note: In this one I used Contains instead of = because in the database there's Sunny 1950s Apartment, St Kilda East
Longer stays
So when I query MATCH (r :Review) – [re :REVIEW] -> (l:Listing {name:'Sunny 1950s Apartment, St Kilda East Longer stays'}) RETURN count(r);

This won't show anything, and from the specification that there is no empty query.

2. Show all reviews in Port Phillip

MATCH (r:Review)–[re:REVIEWS]->(l:Listing) WHERE l.neighbourhood='Port Phillip' RETURN r as review, l.neighbourhood as location;

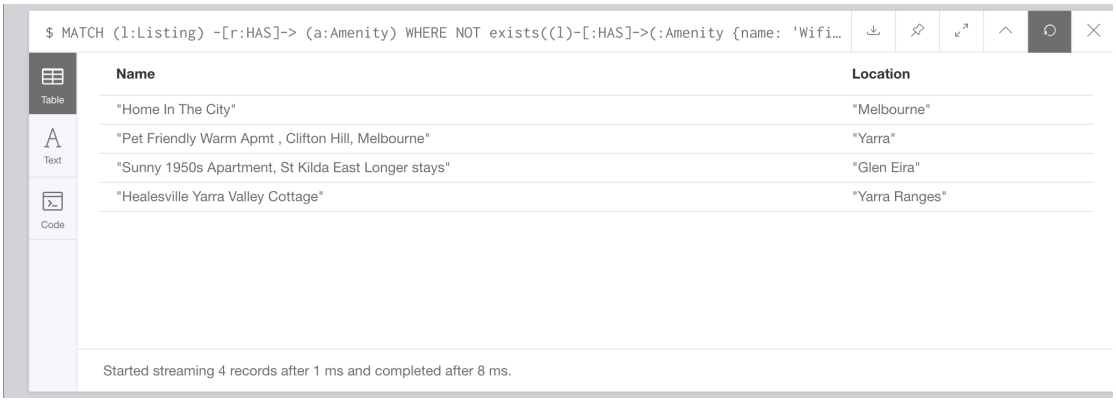


3. Can you recommend accommodations that Jerome (reviewer 4162110) has never been but Sandy & Pete (reviewer 317848) have stayed and gave ratings above 90?

MATCH(r:Review) -[re:REVIEWS]->(l:Listing), (rev:Reviewer)-- (r:Review) WHERE (rev.reviewerId ='317848') AND (rev.reviewerId<>'4162110') AND (re.reviewScoresRating>=90) RETURN l.name as accommodations;

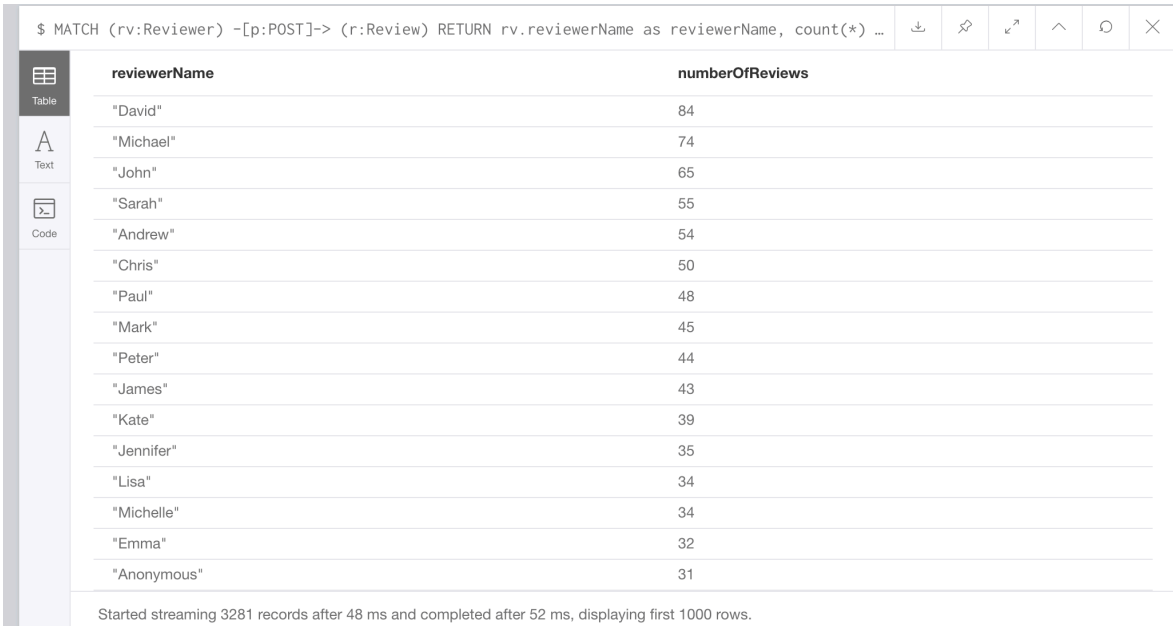
4. List all accommodation names and locations that do not provide Wi-Fi.

MATCH (l:Listing) -[r:HAS]-> (a:Amenity) WHERE NOT exists((l)-[:HAS]->(:Amenity {name:'Wifi'})) RETURN DISTINCT l.name as Name, l.neighbourhood as Location;



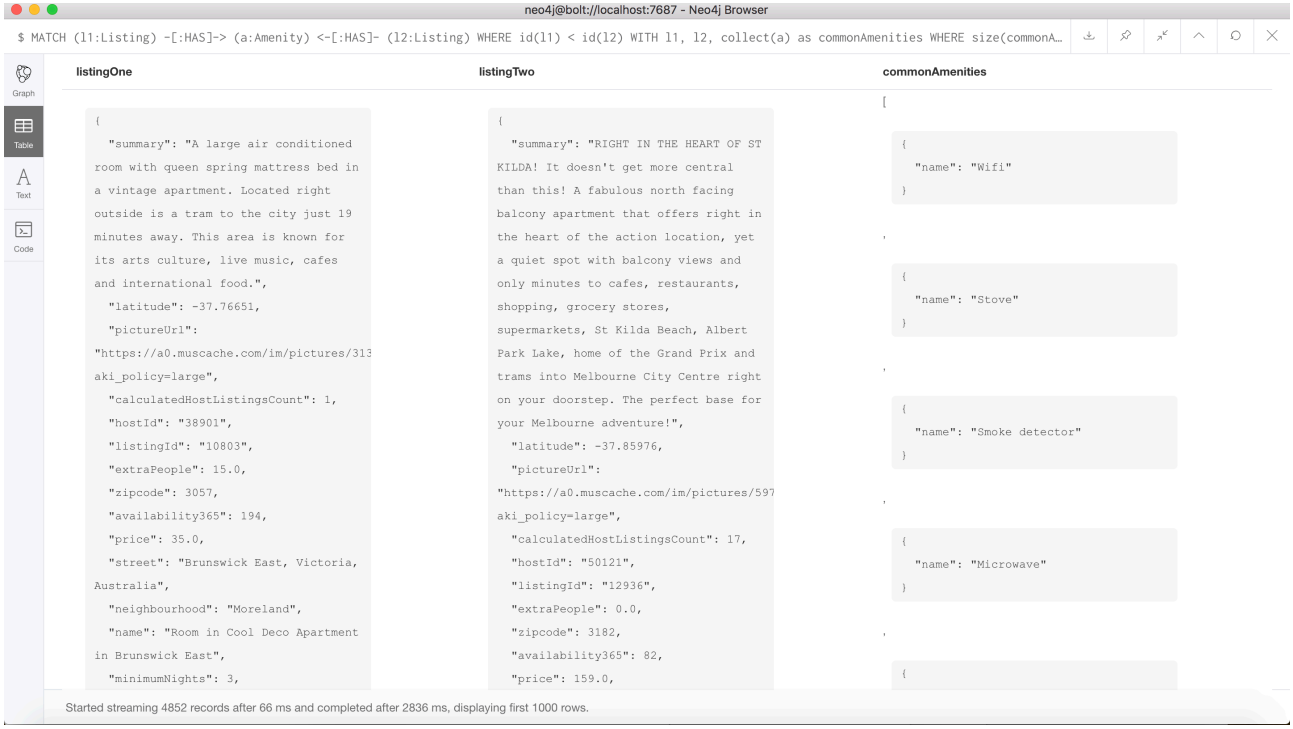
5.Count how many times a reviewer left reviews.
MATCH (rv:Reviewer) -[p:POST]-> (r:Review) RETURN rv.reviewerName as reviewerName, count(*) as numberOfReviews ORDER by numberOfReviews DESC;
Or
MATCH (rv:Reviewer) -[p:POST]-> (r:Review) RETURN rv.reviewerName as reviewerName, count(p) as numberOfReviews ORDER by numberOfReviews DESC;

Both shows the same result



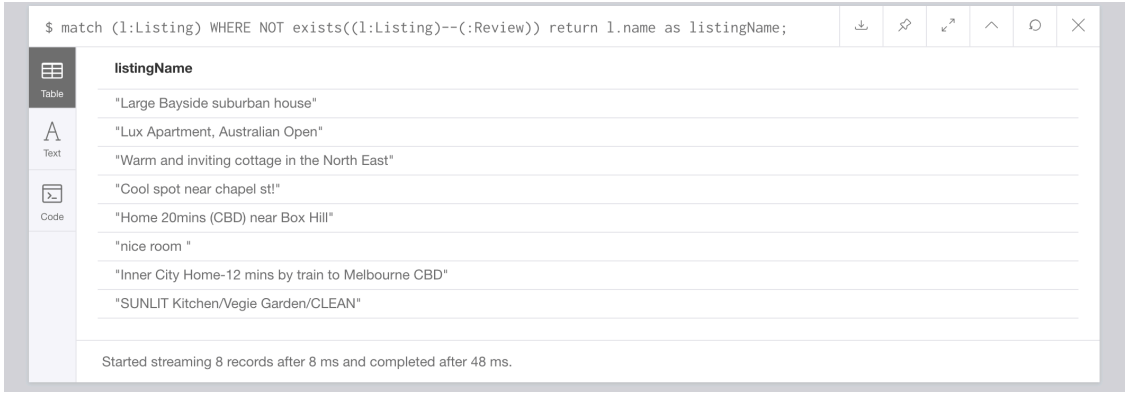
6. Display a list of pairs of accommodations having more than three amenities in common.

```
MATCH (l1:Listing) -[:HAS]-> (a:Amenity) <-[:HAS]- (l2:Listing)
WHERE id(l1) < id(l2)
WITH l1, l2, collect(a) as commonAmenities
WHERE size(commonAmenities) > 3
RETURN l1 as listingOne, l2 as listingTwo, commonAmenities;
```



7. Which listings do not have any review?

```
match (l:Listing) WHERE NOT exists((l:Listing)--(:Review)) return l.name as listingName;
```



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

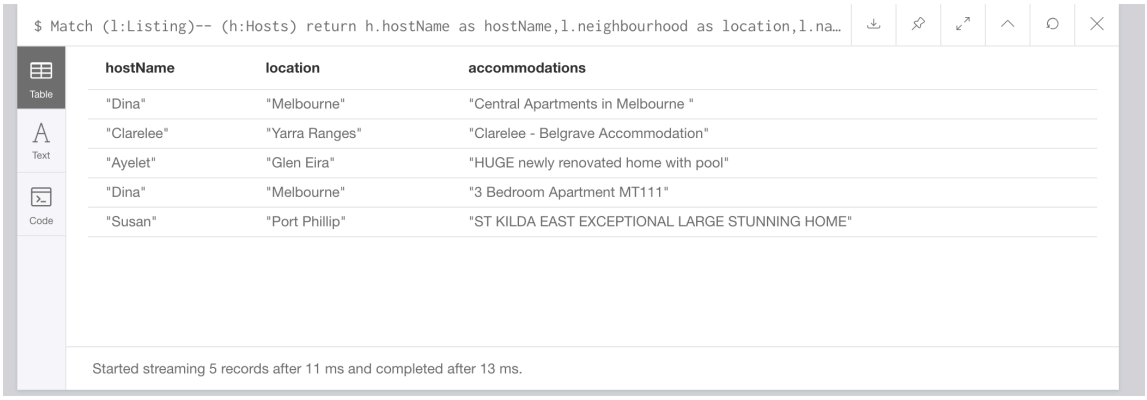
match (h:Hosts)-[:HOSTS]-> (l:Listing) with h,count(l) as cnt, collect(l.name) as AccomedationName,collect(l.price) as Price WHERE cnt > 1 Return h.hostName as hostName, AccomedationName,Price;

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

Match (l:Listing) WHERE l.neighbourhood='Melbourne' return avg(l.price) as averagePrice;

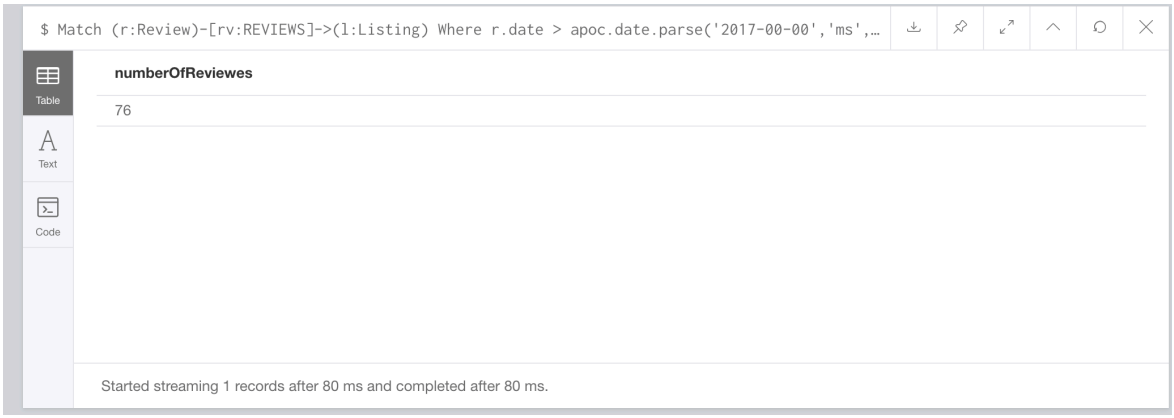
10 - Where are the top 5 most expensive accommodations? Display the locations, host information, and names of those accommodation

Match (l:Listing)-- (h:Hosts) return h.hostName as hostName,l.neighbourhood as location,l.name as accommodations ORDER BY l.price DESC LIMIT 5;



11. How many accommodations were reviewed in 2017?

Match (r:Review)-[rv:REVIEWS]->(l:Listing) Where r.date > apoc.date.parse('2017-00-00','ms','yyyy-MM-dd') AND r.date < apoc.date.parse('2018-00-00','ms','yyyy-MM-dd') Return count(DISTINCT l.name) as numberOfReviewes;

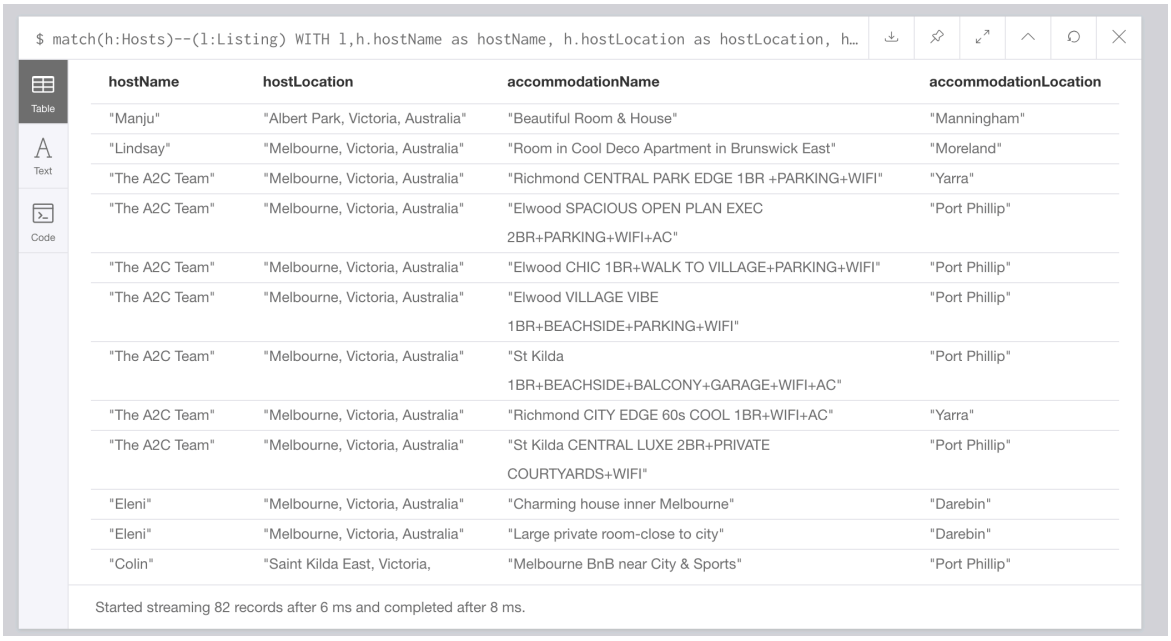


12. What are the top 10 most popular neighbourhoods based on the total average review score ratings?

MATCH (r:Review)-[rv:REVIEWS]->(l:Listing) with l.neighbourhood as location ,avg(rv.reviewScoresRating) as avg return location ORDER BY avg DESC LIMIT 10;

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

match(h:Hosts)--(l:Listing) WITH l,h.hostName as hostName, h.hostLocation as location, h.hostLocation CONTAINS l.neighbourhood as neg where neg= false return DISTINCT hostName,hostLocation,l.name as accommodationName ,l.neighbourhood as accommodationLocation;

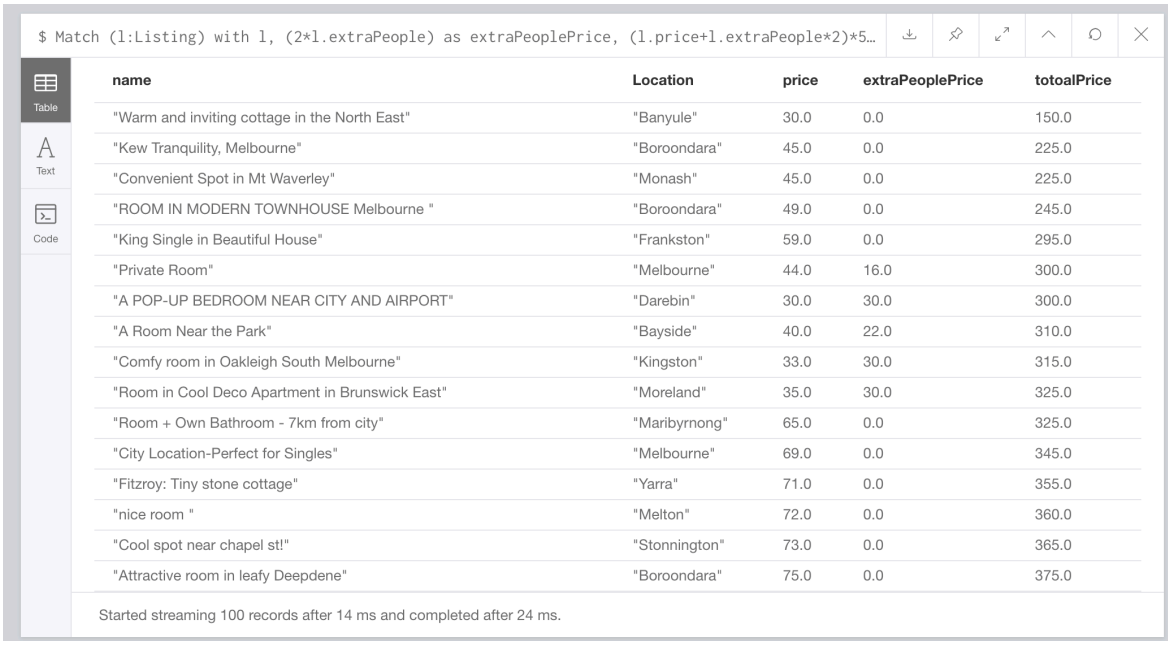


hostName	hostLocation	accommodationName	accommodationLocation
"Manju"	"Albert Park, Victoria, Australia"	"Beautiful Room & House"	"Manningham"
"Lindsay"	"Melbourne, Victoria, Australia"	"Room in Cool Deco Apartment in Brunswick East"	"Moreland"
"The A2C Team"	"Melbourne, Victoria, Australia"	"Richmond CENTRAL PARK EDGE 1BR +PARKING+WIFI"	"Yarra"
"The A2C Team"	"Melbourne, Victoria, Australia"	"Elwood SPACIOUS OPEN PLAN EXEC 2BR+PARKING+WIFI+AC"	"Port Phillip"
"The A2C Team"	"Melbourne, Victoria, Australia"	"Elwood CHIC 1BR+WALK TO VILLAGE+PARKING+WIFI"	"Port Phillip"
"The A2C Team"	"Melbourne, Victoria, Australia"	"Elwood VILLAGE VIBE 1BR+BEACHSIDE+PARKING+WIFI"	"Port Phillip"
"The A2C Team"	"Melbourne, Victoria, Australia"	"St Kilda 1BR+BEACHSIDE+BALCONY+GARAGE+WIFI+AC"	"Port Phillip"
"The A2C Team"	"Melbourne, Victoria, Australia"	"Richmond CITY EDGE 60s COOL 1BR+WIFI+AC"	"Yarra"
"The A2C Team"	"Melbourne, Victoria, Australia"	"St Kilda CENTRAL LUXE 2BR+PRIVATE COURTYARDS+WIFI"	"Port Phillip"
"Eleni"	"Melbourne, Victoria, Australia"	"Charming house inner Melbourne"	"Darebin"
"Eleni"	"Melbourne, Victoria, Australia"	"Large private room-close to city"	"Darebin"
"Colin"	"Saint Kilda East, Victoria,	"Melbourne BnB near City & Sports"	"Port Phillip"

Started streaming 82 records after 6 ms and completed after 8 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:Listing) with l, (2*l.extraPeople) as extraPeoplePrice, (l.price+l.extraPeople*2)*5 as totoalPrice return l.name as name ,l.neighbourhood as Location ,l.price as price, extraPeoplePrice,totoalPrice ORDER BY totoalPrice;



name	Location	price	extraPeoplePrice	totoalPrice
"Warm and inviting cottage in the North East"	"Banyule"	30.0	0.0	150.0
"Kew Tranquility, Melbourne"	"Boroondara"	45.0	0.0	225.0
"Convenient Spot in Mt Waverley"	"Monash"	45.0	0.0	225.0
"ROOM IN MODERN TOWNHOUSE Melbourne "	"Boroondara"	49.0	0.0	245.0
"King Single in Beautiful House"	"Frankston"	59.0	0.0	295.0
"Private Room"	"Melbourne"	44.0	16.0	300.0
"A POP-UP BEDROOM NEAR CITY AND AIRPORT"	"Darebin"	30.0	30.0	300.0
"A Room Near the Park"	"Bayside"	40.0	22.0	310.0
"Comfy room in Oakleigh South Melbourne"	"Kingston"	33.0	30.0	315.0
"Room in Cool Deco Apartment in Brunswick East"	"Moreland"	35.0	30.0	325.0
"Room + Own Bathroom - 7km from city"	"Maribyrnong"	65.0	0.0	325.0
"City Location-Perfect for Singles"	"Melbourne"	69.0	0.0	345.0
"Fitzroy: Tiny stone cottage"	"Yarra"	71.0	0.0	355.0
"nice room "	"Melton"	72.0	0.0	360.0
"Cool spot near chapel st!"	"Stonnington"	73.0	0.0	365.0
"Attractive room in leafy Deepdene"	"Boroondara"	75.0	0.0	375.0

Started streaming 100 records after 14 ms and completed after 24 ms.

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:Listing),(ls:Listing) WHERE l<>ls RETURN distance(point({latitude: l.latitude, longitude: l.longitude}), point({latitude: ls.latitude, longitude: ls.longitude})) / 1000.0 as distance,l as listingOne,ls as listingTwo ORDER BY distance ASC LIMIT 10;

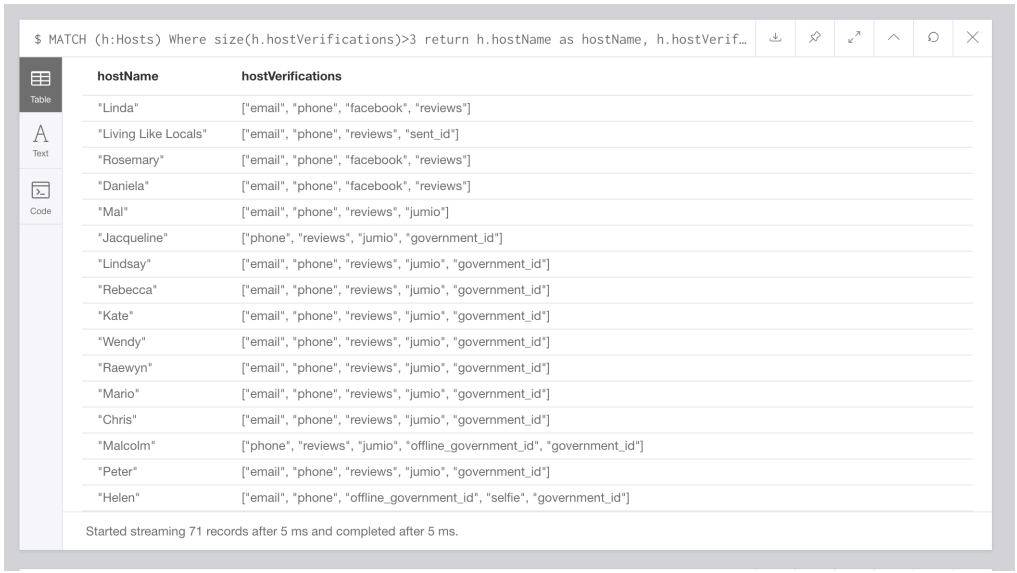
Five additional queries:

16. Show accommodations who have a 'Free parking on premises' amenity and the price per night is less than 100 and room type is 'Entire home/apt' return name and price

MATCH (l:Listing) -[r:HAS]-> (a:Amenity) WHERE exists((l)-[:HAS]->(:Amenity {name:'Free parking on premises'})) AND l.roomType='Entire home/apt' AND l.price<100 RETURN DISTINCT l.name as accommodation, l.price as price;

17- Lists hosts who has more than three host verifications. List host name and host verifications.

MATCH (h:Hosts) Where size(h.hostVerifications)>3 return h.hostName as hostName, h.hostVerifications as hostVerifications ORDER BY size(hostVerifications);



$ MATCH (h:Hosts) Where size(h.hostVerifications)>3 return h.hostName as hostName, h.hostVerif…

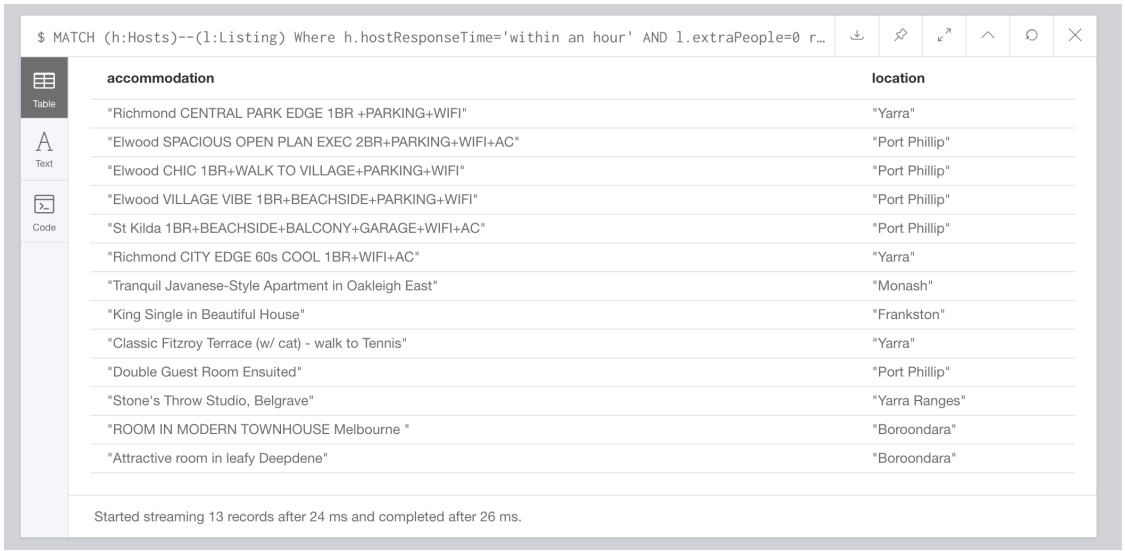
Table | Text | Code

hostName	hostVerifications
"Linda"	["email", "phone", "facebook", "reviews"]
"Living Like Locals"	["email", "phone", "reviews", "sent_id"]
"Rosemary"	["email", "phone", "facebook", "reviews"]
"Daniela"	["email", "phone", "facebook", "reviews"]
"Mal"	["email", "phone", "reviews", "jumio"]
"Jacqueline"	["phone", "reviews", "jumio", "government_id"]
"Lindsay"	["email", "phone", "reviews", "jumio", "government_id"]
"Rebecca"	["email", "phone", "reviews", "jumio", "government_id"]
"Kate"	["email", "phone", "reviews", "jumio", "government_id"]
"Wendy"	["email", "phone", "reviews", "jumio", "government_id"]
"Raewyn"	["email", "phone", "reviews", "jumio", "government_id"]
"Mario"	["email", "phone", "reviews", "jumio", "government_id"]
"Chris"	["email", "phone", "reviews", "jumio", "government_id"]
"Malcolm"	["phone", "reviews", "jumio", "offline_government_id", "government_id"]
"Peter"	["email", "phone", "reviews", "jumio", "government_id"]
"Helen"	["email", "phone", "offline_government_id", "selfie", "government_id"]

Started streaming 71 records after 5 ms and completed after 5 ms.

18- Display listing who has host who response ‘within an hour’ and no price for extra people. Display the listing name and neighbour.

MATCH (h:Hosts)--(l:Listing) Where h.hostResponseTime='within an hour' AND l.extraPeople=0 return l.name as accommodation,l.neighbourhood as location;



$ MATCH (h:Hosts)--(l:Listing) Where h.hostResponseTime='within an hour' AND l.extraPeople=0 r…

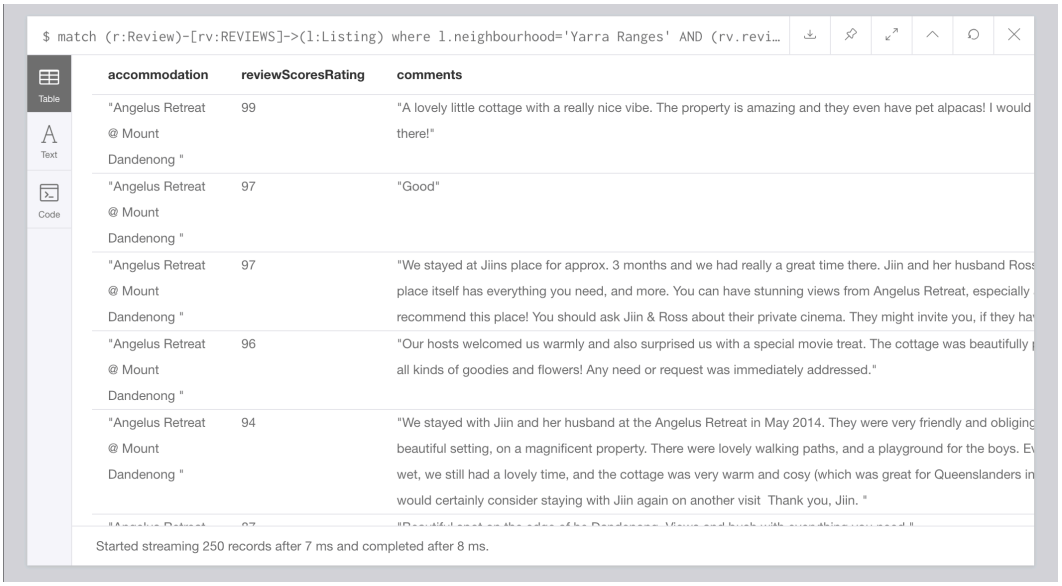
Table | Text | Code

accommodation	location
"Richmond CENTRAL PARK EDGE 1BR +PARKING+WIFI"	"Yarra"
"Elwood SPACIOUS OPEN PLAN EXEC 2BR+PARKING+WIFI+AC"	"Port Phillip"
"Elwood CHIC 1BR+WALK TO VILLAGE+PARKING+WIFI"	"Port Phillip"
"Elwood VILLAGE VIBE 1BR+BEACHSIDE+PARKING+WIFI"	"Port Phillip"
"St Kilda 1BR+BEACHSIDE+BALCONY+GARAGE+WIFI+AC"	"Port Phillip"
"Richmond CITY EDGE 60s COOL 1BR+WIFI+AC"	"Yarra"
"Tranquil Javanese-Style Apartment in Oakleigh East"	"Monash"
"King Single in Beautiful House"	"Frankston"
"Classic Fitzroy Terrace (w/ cat) - walk to Tennis"	"Yarra"
"Double Guest Room Ensuited"	"Port Phillip"
"Stone's Throw Studio, Belgrave"	"Yarra Ranges"
"ROOM IN MODERN TOWNHOUSE Melbourne "	"Boroondara"
"Attractive room in leafy Deepdene"	"Boroondara"

Started streaming 13 records after 24 ms and completed after 26 ms.

19- Display listing in Yarra ranges who has a review score of 85 and more and has been reviewed in 2019, show the listing name, score, and comments. Rank the accomodations by name and score rating from highest to lowest.

match (r:Review)-[rv:REVIEWS]->(l:Listing) where l.neighbourhood='Yarra Ranges' AND (rv.reviewScoresRating>=85) return l.name as accommodation,rv.reviewScoresRating as reviewScoresRating ,r.comments as comments ORDER BY accommodation,reviewScoresRating DESC;



20- Show the 10 last review on the accomodation ‘Northcote, A Classic in Melbourne’ and list them with highest score.

match (r:Review)-[rv:REVIEWS]->(l:Listing) where l.name='Northcote, A Classic in Melbourne' AND rv.reviewScoresRating IS NOT NULL return l.name as accommodation, rv.reviewScoresRating as reviewScoresRating, apoc.date.format(r.date,'ms','yyyy-MM-dd') as reviewDate order by reviewDate DESC LIMIT 10;

Indices:

For creating indices, I created the indices on the most used fields in the above queries:
Single:

CREATE INDEX ON :Listing(neighbourhood);

Composite:

CREATE INDEX ON :Listing(neighbourhood,name);

CREATE INDEX ON :Hosts(hostName,hostLocation);

C.3. Database Modifications.

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.**

In this task only one listing has been provided here with screen shot, other two listing will be in the TaskC3.cypher

Create Listing:
CREATE (l:Listing {listingId: "455543", listingName: "The loft, Villa Maria Circa 1890", summary: "*Villa Maria Beaconsfield Circa 1890 This charming old homestead and country chapel, 100 metres from the Old Princess Hwy (train station 13 min walk, Monash Fwy close by) is ideally located on the gateway to Gippsland. This open aired apartment is craftsman built, detailed and has architectural shaped ceilings. A beautiful relaxing space, which has its own parking, private entry foyer and is locked separately form the main house. Situated on a rise, in a quiet court with open undulating views.", listingUrl:"https://www.airbnb.com.au/rooms/35546338?source_impression_id=p3_1571800887_mShQQ8gVMW8%2FwnEV", listingPictureUrl: "https://a0.muscache.com/im/pictures/2b79afba-bef1-4cea-af17-dbc21ffa75c5.jpg?aki_policy=xx_large", hostId: "164193", neighbourhood: "Beaconsfield", street: " Beaconsfield, Victoria, Australia", zipcode: 3807, latitude: -38.040713, longitude: 145.378137, roomType: "Entire home", price: 120.00, extraPeople: 0.00, minimumNights: 1, calculatedHostListingsCount: 4, availability365: 20});

Create Amenities for the listing:

MATCH (l:Listing),(a:Amenity)
WHERE l.listingId="455543" AND a.name="Heating"
CREATE (l)-[:HAS]-> (a);

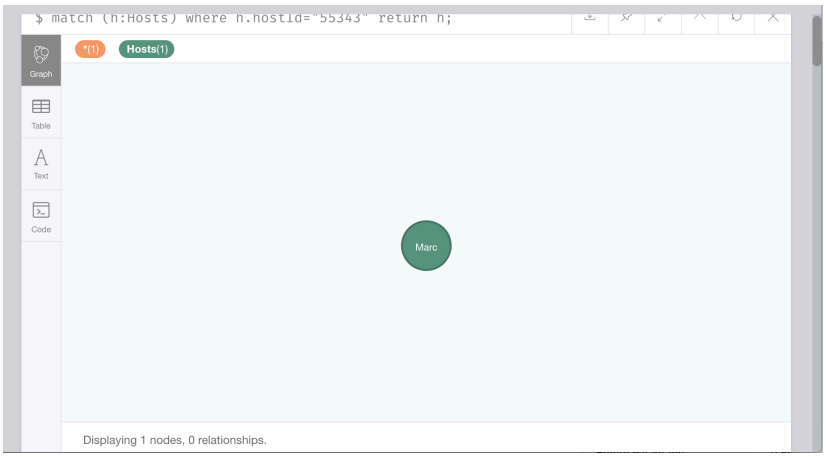
MATCH(l:Listing),(a:Amenity)
WHERE l.listingId="455543" AND a.name="Hot water"
CREATE(l)-[h:HAS]->(a);

MATCH (l:Listing), (a:Amenity)
WHERE l.listingId = '455543' AND a.name = 'TV'
CREATE (l) -[:HAS]-> (a);

MATCH (l:Listing), (a:Amenity)
WHERE l.listingId = '455543' AND a.name = 'Air conditioning'
CREATE (l) -[:HAS]-> (a);

Create Host:

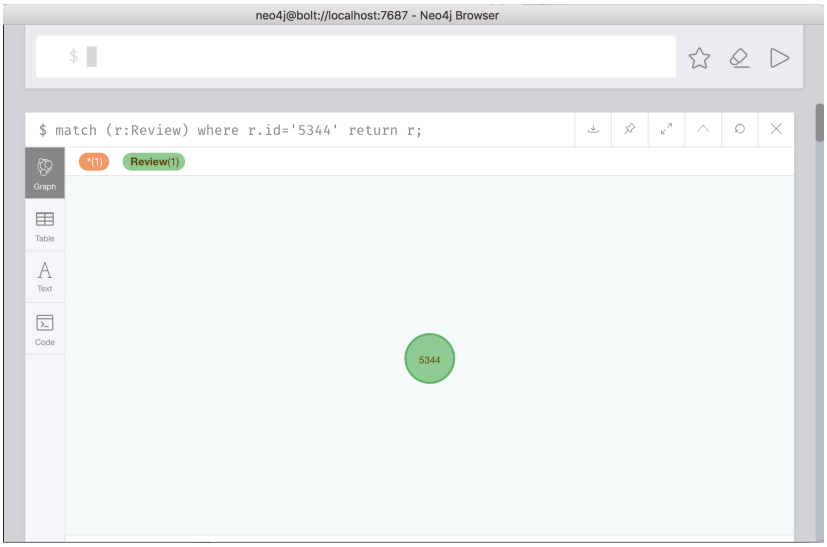
CREATE (h:Hosts {hostId: "55343", hostUrl: "https://www.airbnb.com.au/users/show/267444171", hostName: "Marc", hostVerifications: apoc.convert.fromJsonList("['email', 'phone', 'selfie', 'government_id']"), hostSince: apoc.date.parse("2019-06-21", 'ms','yyyy-MM-dd'), hostLocation: "Beaconsfield, Victoria, Australia", hostResponseTime: "within an hour", hostIsSuperhost: true});



Creating Relationship between host and listing:
MATCH (h:Hosts), (l:Listing)
WHERE l.listingId = '455543' AND h.hostId = '55343'
CREATE (h) -[:HOSTS]-> (l);
Creating Review:

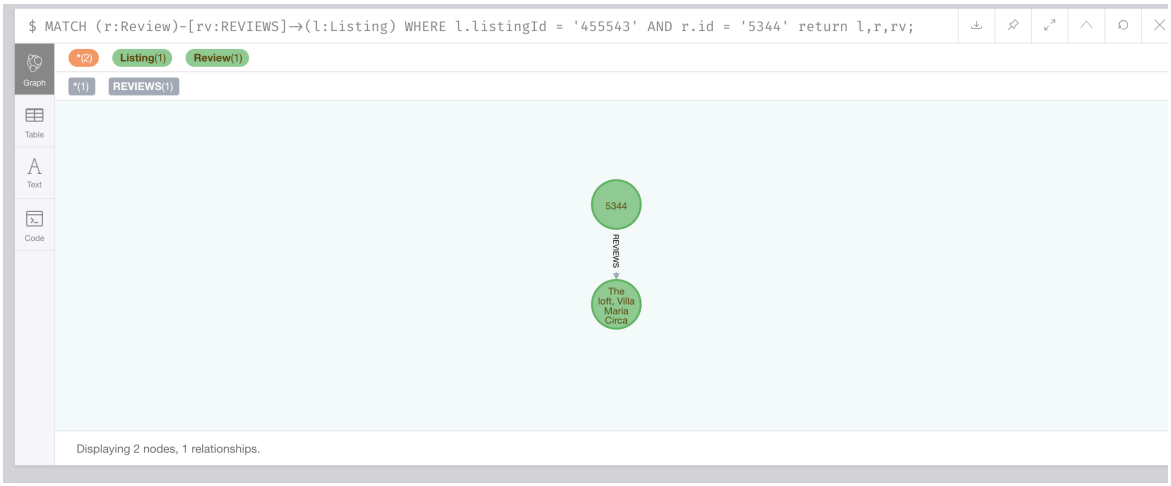
Create Review:

CREATE (r:Review {id: "5344", listingId: "455543", date: apoc.date.parse("2019-10-10", 'ms','yyyy-MM-dd'), comments: "Beautifully built loft. Was perfectly cleaned and presented and had everything I needed. Easy access to wifi and a great check in/out experience. Can't wait to stay there again the next time I am in town."});

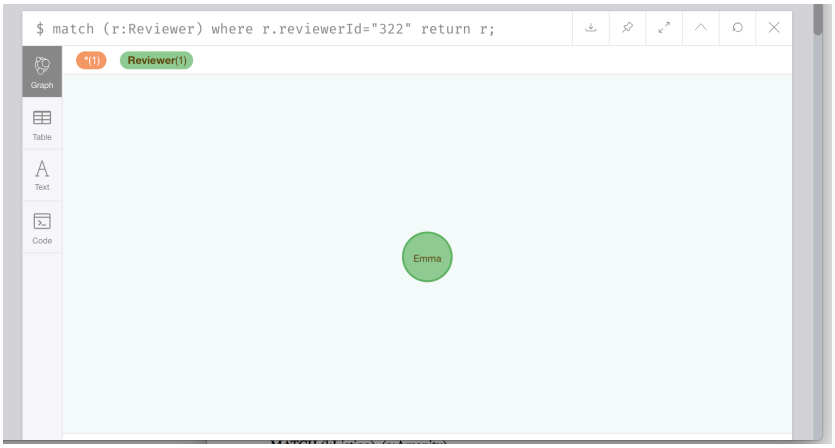


Create relationship between review and listing:

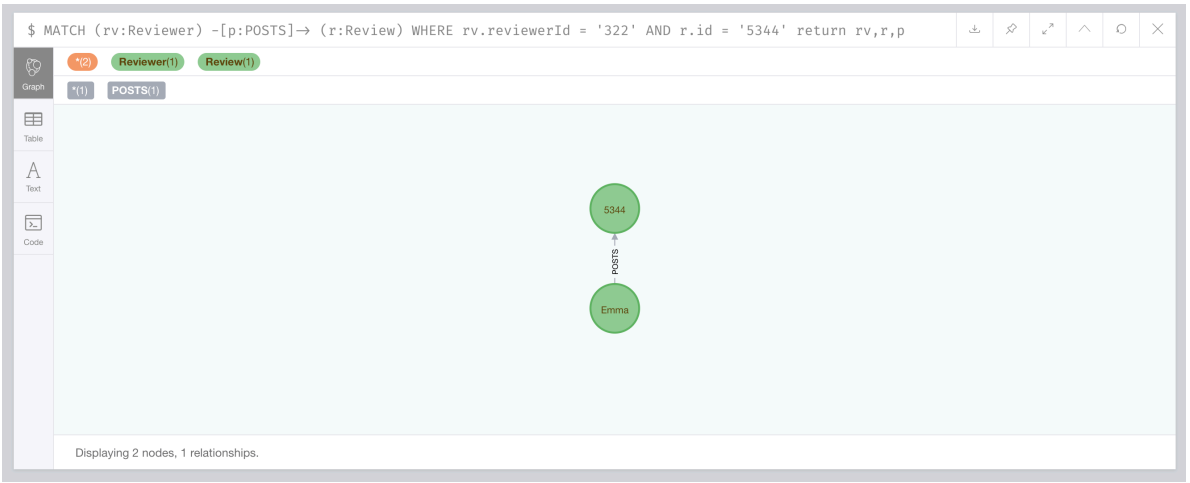
MATCH (l:Listing), (r:Review)
WHERE l.listingId = '455543' AND r.id = '5344'
CREATE (r) -[rv:REVIEWS{reviewScoresRating:87}]-> (l);



Create Reviewer:
CREATE (rvr:Reviewer {reviewerId: "322", reviewerName: “Emma"});



Create relationship between review and reviewer:
MATCH (rv:Reviewer), (r:Review)
WHERE rv.reviewerId = '322' AND r.id = '5344'
CREATE (rv) -[:POSTS]-> (r);



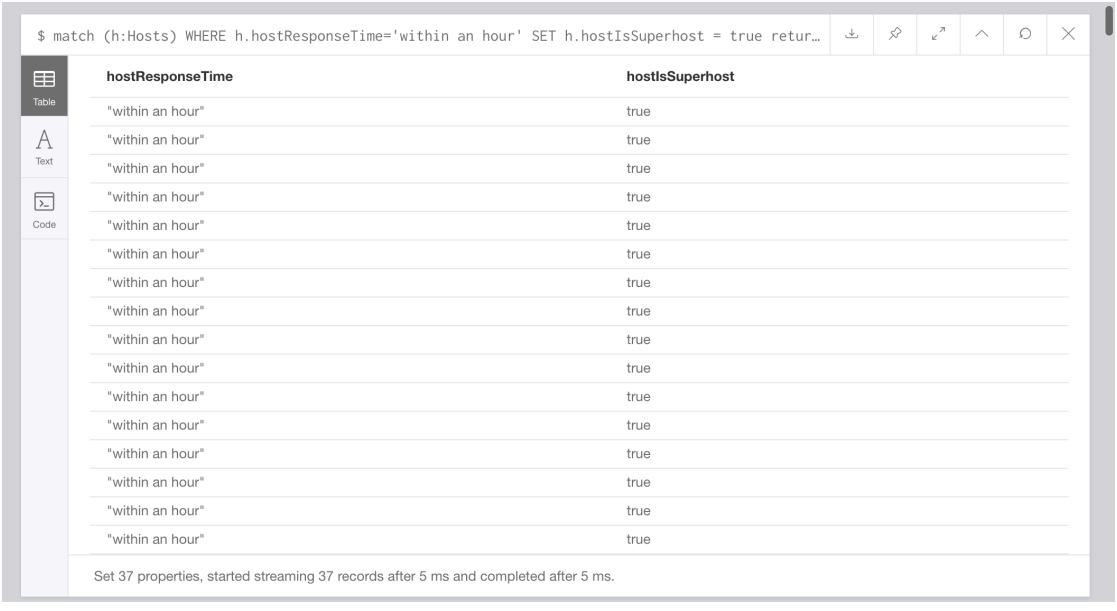
2. Update the host verification for those who registered in 2009 and add Facebook to the list of existing verifications.

This query will add facebook to the hostVerifcation of the hosts who registered in 2009 and doesn't have facebook, so will add the new host verification which is facebook. Because, if the facebook already exit no need to add it again.

match (h:Hosts) where h.hostSince> apoc.date.parse('2009-00-00','ms','yyyy-MM-dd') AND h.hostSince < apoc.date.parse('2010-00-0','ms','yyyy-MM-dd') AND not "facebook" in h.hostVerifications SET h.hostVerifications= h.hostVerifications +"facebook" return h.hostName as hostName,h.hostVerifications as hostVerifications ,apoc.date.format(h.hostSince,'ms','yyyy-MM-dd') as date;

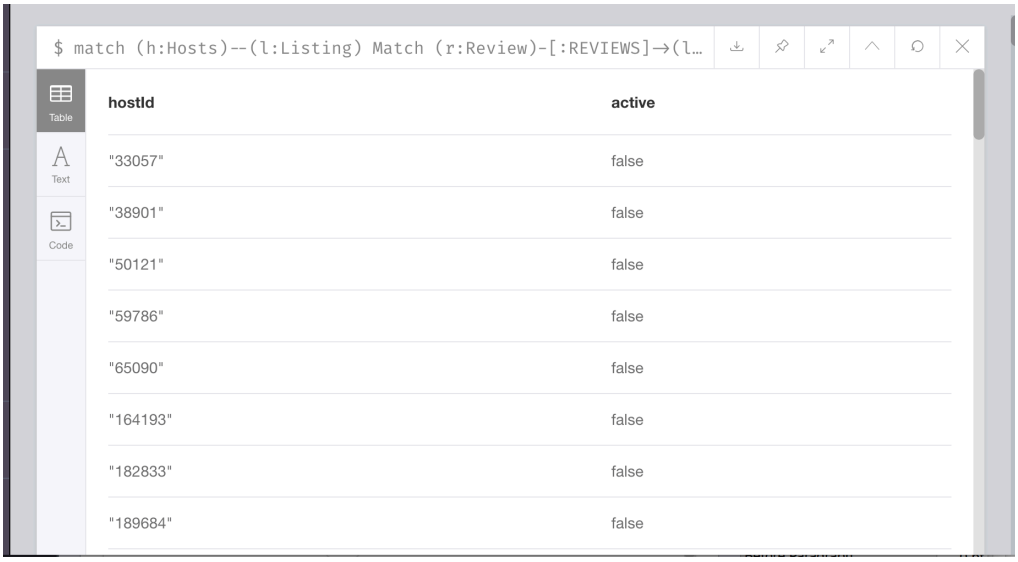
3. Update hosts who respond "within an hour" to a superhost. For this update you may only use the "host response time" and "host is a super host" information

match (h:Hosts) WHERE h.hostResponseTime='within an hour' SET h.hostIsSuperhost = true return h.hostResponseTime as hostResponseTime , h.hostIsSuperhost as hostIsSuperhost;



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)--(l:Listing) Match (r:Review)-[:REVIEWS]->(l) where r.date <apoc.date.parse(‘2017-00-00','ms','yyyy-MM-dd') SET h.active=(case h.active when 't' then true else false end)
return DISTINCT h.hostId as hostId,h.active as active;



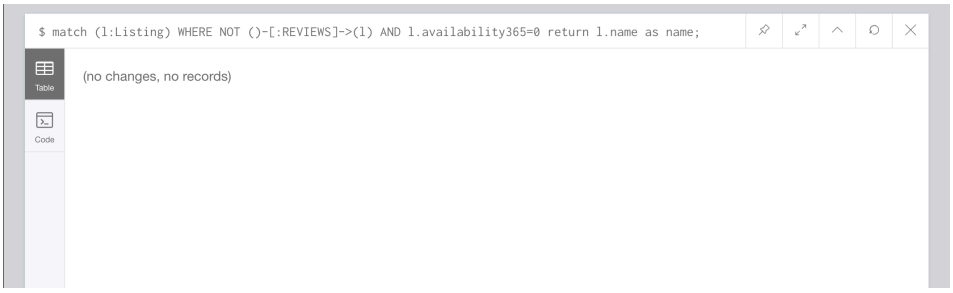
5. Delete all listings with zero availability and have no reviews.

match (l:Listing) WHERE NOT ()-[:REVIEWS]->(l) AND l.availability365=0 return l.name as name;

Then after running this query, the listing will be deleted:

match (l:Listing) WHERE NOT ()-[:REVIEW]->(l) AND l.availability365=0 DETACH DELETE l;

Then I run the first query again, and now it shows nothing:



C.4. Advanced Topic.

Option 2:

Neo4j is used on many websites such as Airbnb, and the reason is that these kinds of websites have a huge amount of data where their old database couldn't handle it. The retrieving of data takes a lot of time and effort for the Airbnb employee; thus, they have to spend a lot of time before if they want to query a huge amount of data. Therefore, they have used a graph database where they can see the nodes as a graph and it makes it easier for them to know what they want to query and how. Using neo4j improves scalability and this what companies aim for. In Airbnb, they aim to address the Democratizing Data problem, to find a solution for this problem to help their employee data exploration and discovery. The uses a graph database, which is easier to see the nodes as a graph and show the relationship. The reason is that, in websites like Airbnb, To handle a growing volume of connected data. They have used Dataportal, which is an online tool that helps with data discovery and decision-making at Airbnb. Dataportal is used to solve the issue with the proliferation of tribal knowledge, which stifles productivity. This is an overall idea of how neo4j can be used and how Airbnb used it in their real-life. The best way to maintain travel and hospitality data is to use graph, because the graph shows the connections between differs nodes and in the hospitality data is the locations of the users and the accommodations, for example, and showing the relationship in a graph and how these two locations(nodes) are connected is the best way for website that used for hospitality. Since MonashBnb is considered to be a hospitality website, we have used neo4j to develop the database. However, the design that has been provided is just a basic design that can be improved more to add more functionalities and make it more efficient. To improve the MonashBnb database, it is better to apply a knowledge graph that can store structured data that connects the several things such as what our client wants, and what the world has to offer for the travelers (places to stay) and to make MonashBnb unique. This suggestion is simply what Airbnb is doing in

its Database, and I suggested the same because I believe Airbnb has one of the best database backbones. This is the suggestion Infrastructure to implement for MonashBnb.

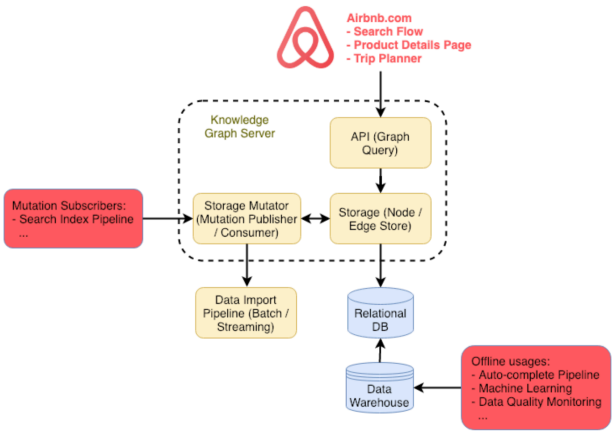


Figure 1: Airbnb Infrastructure from:https:// medium.com/airbnb-engineering/contextualizing-

The above Infrastructure shows how knowledge graph architecture. It consists of three components: graph storage, graph query API, and storage mutator. Graph storage basically stores the local information of the accommodations, which stores the nodes and the relationship (edges) and can directly perform CRUD (create, read, update, and delete) operations. At Airbnb, the information in the graph storage isn't constantly retrieved through online inquiries, so they additionally dump an everyday depiction of the hubs and edges into an information stockroom for disconnected uses, if the connections go offline and that can be implemented on MonashBnb. Auto-complete, for example, rely upon knowledge graph's information dump for their item needs. What's more, we likewise apply for AI advances on the information dump for purposes including chart installing, learning deduction, and so on. All theses that have been used in Airbnb can be used on MonashBnb to improve it.

Besides that, Airbnb has used Taxonomy to Categorise the World of Travel, and I believe this feature is one of the important features that should be implemented on MonashBnb and improve its performance. Additionally, neo4j allows installing of plugging and API which can be done on MonashBnb to maybe improve the type of stored data and get the benefits of these plugins to improve the scalability and performance of querying. In MonashBnb, only APOC has been used to deal with date and array data type because of the lake data type provided originally by neo4j.

References:

https://search-proquest-com.ezproxy.lib.monash.edu.au/docview/2134282591/fulltextPDF/E069951576A341D0PQ/1?accountid=12528

https://neo4j.com/blog/democratizing-data-discovery-airbnb/

https://medium.com/airbnb-engineering/contextualizing-airbnb-by-building-knowledge-graph-b7077e268d5a