

1.1 Insert Property

This query is divided in 3 phases :

1. Fetch max property id
2. Insert the new property with fetched id + 1 in phases 1
3. Take care of consequential edge creations.

Phase 1 :

```
LET property_id = (
    FOR p in property
    FILTER LEFT(p._key,6) == "69123:"
    LET property_id_value = TO_NUMBER(SUBSTRING(p._key,6,3))
    SORT property_id_value DESC
    LIMIT 1
    return TO_NUMBER(SUBSTRING(p._key,6,3))
)
```

Here we iterate over collection property with filter over key such that left 6 characters in key string have

"69123:". From such key we extract digits after ":" and sort them in descending order and limit it to 1 to get max.

Phase 2 :

```
LET new_property_id = property_id + 1
```

```
LET data = [
```

```
{
  "_key": CONCAT("69123:",new_property_id),
  "property_info": {
    "advertise": {
      "property_title": "3.5 bhk",
      "type_of_deal": "rent",
      "minimum_duration_of_stay": 3,
      "available_from_date": "2019-01-09",
      "available_to_date": "2019-04-09",
      "sizeof_property": 15
    },
    "property_status": "available",
    "property_address": {
      "street_name": "Mariaprobst Strasse 3",
      "flat_number": "1105",
      "city": "Heidelberg",
      "pincode": "69123",
      "latitude": 35.1269874,
      "longitude": 4534897523
    },
    "amount": {
      "base_price": 400,
      "nebenkosten": 150,
      "deposit_amount": 700,
      "cost_per_square_meter": 0
    }
  }
}]
```

```
FOR d IN data
  INSERT d INTO property
```

Once we get max key id we increment it by one and insert it along with new property.

Phase 3 :

```
INSERT {
  "_from": CONCAT("property/69123:",new_property_id),
  "_to": "property/69123",
  "relationship": "in"
}
```

```

} INTO property_feature_mapping

LET user_id = (
  for r in notification
  filter r._to == "property/69123"
  return r._from
)

FOR u in user_id
  INSERT {
    "_from": "property/69123",
    "_to": u,
    "notification_of": CONCAT("property/69123:", new_property_id)
  }
  INTO notification

```

There are multiple consequential relations which need to be created once a new property is inserted. Here we demonstrate one out of them. In this case as the property belongs to pincode 69123.

Therefore we insert first create a new edge in collection `property_feature_mapping` with `from` as new property id and `to` as the feature pincode 69123 with edge description as `in`.

later we retrieve user subscribe to pin code 69123. Here we query the edge collection `notification` where the filter is PIN code.

Now once we have the user IDS whom we have to notify we just create a new edge in `notification` where the `from` will be the PIN code and `to` would be the user id with a edge description as the new property ID.

1.2 User Insert

2.1 Retrieve

```

for p in property
filter p.advertise.property_title == "3.5 bhk" and p.advertise.type_of_deal == "rent"
and p.amount.base_price <=550 and p.amount.base_price >=350
and p.property_address.city == 'Heidelberg'
and p.advertise.available_from_date >= "2019-01-08"
and p.property_status == "available"
SORT p.sort_priority.weight DESC
return {key : p._key, weight : p.sort_priority.weight}

```

here we query the collection `property` with the user filters. Later to show the results as per the sort priority we sort them in the descending order of the Priority weight.

Query	
4 elements	65.188 ms
key	weight
69123:1	260
69123:10	140
69123:6	120
69123:11	40

```

[
  {

```

```

    "key": "69123:1",
    "weight": 260
  },
  {
    "key": "69123:10",
    "weight": 140
  },
  {
    "key": "69123:6",
    "weight": 120
  },
  {
    "key": "69123:11",
    "weight": 40
  }
]

```

2.2 Retrieve in a graph

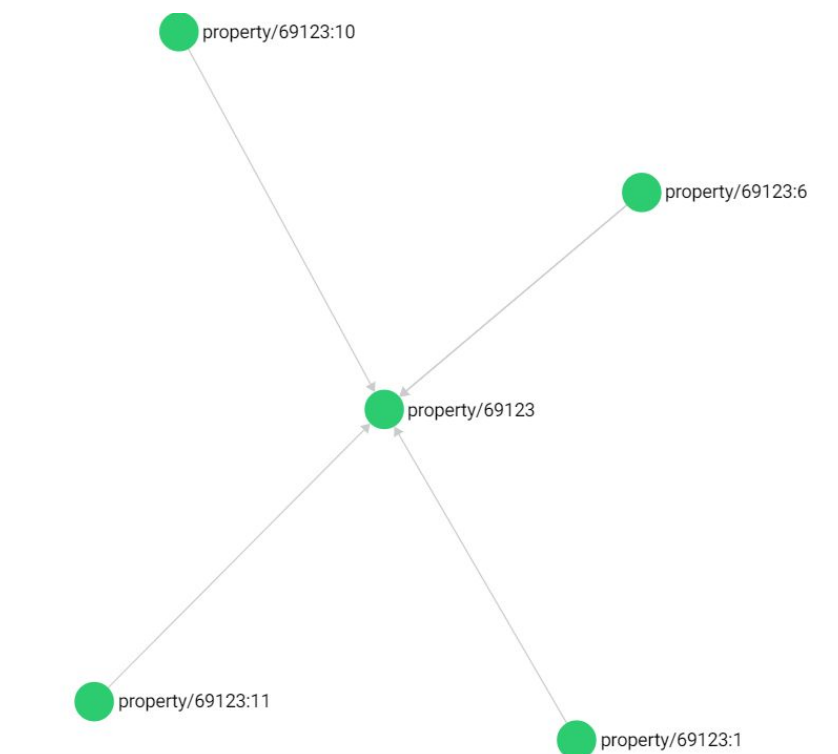
```

let to_show = (
for p in property
filter p.advertise.property_title == "3.5 bhk" and p.advertise.type_of_deal == "rent"
and p.amount.base_price <=550 and p.amount.base_price >=350
and p.property_address.city == 'Heidelberg'
and p.advertise.available_from_date >= "2019-01-08"
and p.property_status == "available"
SORT p.sort_priority.weight DESC
return p._key
)

for t in to_show
  for p in property_feature_mapping
    filter p._to == "property/69123" and p._from == CONCAT("property/",t)
    return p

```

This query can be divided into 2 steps. the first is same as 2.1. using the property ids retrieved we later query the Edge collection property feature mapping with filter as pincode and retrieved property ID in step 1.



```

[
  {
    "_key": "502498",
    "_id": "property_feature_mapping/502498",
    "_from": "property/69123:1",
    "_to": "property/69123",

```

```

    "_rev": "_YJOptly--N",
    "type": "in"
  },
  {
    "_key": "502846",
    "_id": "property_feature_mapping/502846",
    "_from": "property/69123:10",
    "_to": "property/69123",
    "_rev": "_YJOptly--T",
    "type": "in"
  },
  {
    "_key": "502818",
    "_id": "property_feature_mapping/502818",
    "_from": "property/69123:6",
    "_to": "property/69123",
    "_rev": "_YJOptlW--J",
    "type": "in"
  },
  {
    "_key": "502849",
    "_id": "property_feature_mapping/502849",
    "_from": "property/69123:6",
    "_to": "property/69123",
    "_rev": "_YJOptly--Z",
    "type": "in"
  },
  {
    "_key": "502853",
    "_id": "property_feature_mapping/502853",
    "_from": "property/69123:11",
    "_to": "property/69123",
    "_rev": "_YJOptl2--F",
    "type": "in"
  }
]

```

3.1 Stats of prices in a locality

```

let pin = "69123"
let type = "sell"
let all_p = (for d in property
filter d.property_address.pincode == pin and d.advertise.type_of_deal == type
return d.amount.base_price
)
return {Area : pin , "Avg Price" : AVG(all_p),"Min price" : MIN (all_p),"Max Price" : MAX(all_p)}

```

In this query we utilise the apis given to us by arangodb.if we have an array of numbers we can do the aggregation functions on them.

By setting the variable pin and a type of deal with the respective values we query the collection property with these filters and retrieve the base price in a variable all_p. this variable is nothing but an array of base price. while returning we can return the pin and the desired and aggregation function over the price array.

Query 1 elements 111.334 ms				JSON Table
Area	Avg Price	Min price	Max Price	
69123	1175	750	1600	

```

[
  {
    "Area": "69123",
    "Avg Price": 1175,
    "Min price": 750,
    "Max Price": 1600
  }
]

```

3.2 Historical Prices of a Property

```
for p in property
filter p._key == '69123:3'
let h_data = p.stats.historical_prices
for d in h_data
return d
```

In each property document we are storing the historical prices on an annual basis for past 10 years. here we query the collection property with filter as a property key and then retrieve the historical price array and display it.

Query

9 elements

34.038 ms

Date	Price
2010-12	200
2011-12	250
2012-12	380
2013-12	350
2014-12	370
2015-12	400
2016-12	430
2017-12	450
2018-12	480

[

{

"Date": "2010-12",

"Price": 200

}

,

{

"Date": "2011-12",

"Price": 250

}

,

{

"Date": "2012-12",

"Price": 380

}

,

{

"Date": "2013-12",

"Price": 350

}

,

{

"Date": "2014-12",

"Price": 370

}

,

{

"Date": "2015-12",

"Price": 400

}

,

{

"Date": "2016-12",

"Price": 430

}

,

{

"Date": "2017-12",

"Price": 450

}

,

{

"Date": "2018-12",

"Price": 480

}

]

]

3.3 Historical Prices of a region

```
for i in 2010..2019
let str_i = CONCAT(i,"-12")

for p in property
  filter p.stats.historical_prices != null and p.property_address.city == "Heidelberg"

  Collect d = FIRST(p.stats.historical_prices[* filter CURRENT.Date== str_i return CURRENT.Date])
aggregate
  pr = AVG(FIRST(p.stats.historical_prices[* filter CURRENT.Date== str_i return CURRENT.Price])) into g

  filter d !=null and pr != null
  sort d
RETURN {
  Date : d,
  Prices : pr
}
```

As observed in section 3.2 the date format we have is yyyy-mm. also we are storing the historical prices from 2010. using this site to be formulated string for each year and query it were the historical prices of a property. later using the collect clause we do the aggregation of prices on date and get the average price. then to keep the data clean the filter the null dates and the null prices and sort them in descending order.

Date	Prices
2010-12	167.14285714285714
2011-12	200
2012-12	281.42857142857144
2013-12	307.14285714285717
2014-12	332.85714285714283
2015-12	377.14285714285717
2016-12	415.7142857142857
2017-12	464.2857142857143
2018-12	512.8571428571429

```
[
{
  "Date": "2010-12",
  "Prices": 167.14285714285714
},
{
  "Date": "2011-12",
  "Prices": 200
},
{
  "Date": "2012-12",
  "Prices": 281.42857142857144
},
{
  "Date": "2013-12",
  "Prices": 307.14285714285717
},
{
```

```

    "Date": "2014-12",
    "Prices": 332.85714285714283
  },
  {
    "Date": "2015-12",
    "Prices": 377.14285714285717
  },
  {
    "Date": "2016-12",
    "Prices": 415.7142857142857
  },
  {
    "Date": "2017-12",
    "Prices": 464.2857142857143
  },
  {
    "Date": "2018-12",
    "Prices": 512.8571428571429
  }
]

```

4.1 get all properties recommended to a user

```

let user_id = FIRST(for u in User filter u.user_info.first_name == "paneesh" return u._id)

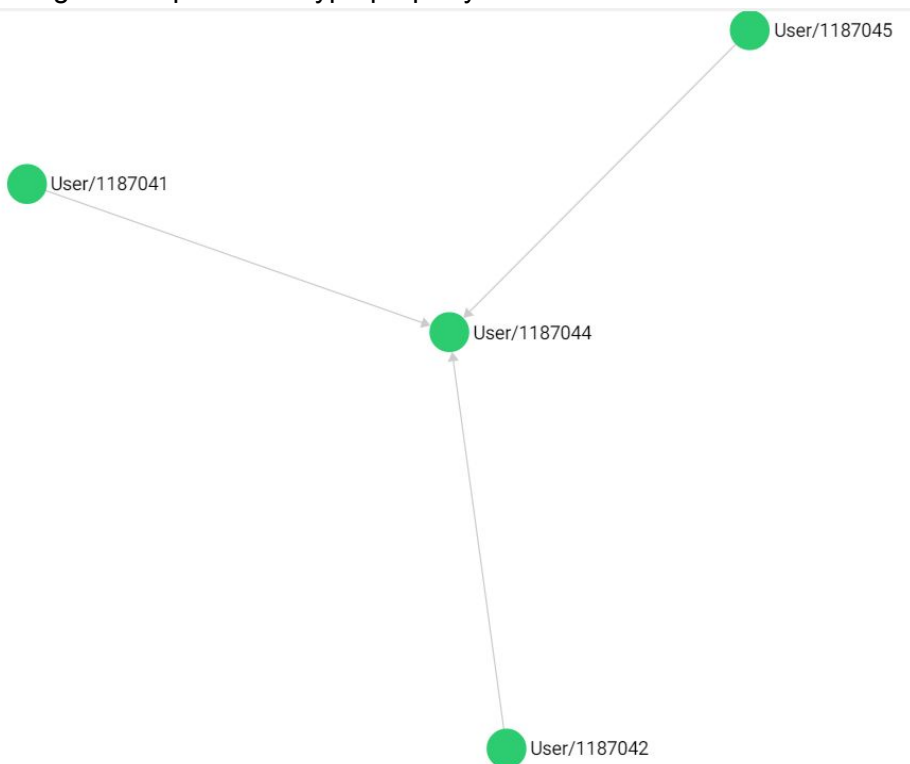
for r in recommend
filter r._to == user_id AND CONTAINS (r.recommended_property,"property/")
return r

```

Querying Edge collection can be done in two ways. here we do with the traditional way as for document.

Forgetting all the properties recommended to user we first need his user id. we do that by querying the collection user with filter first name as the username.

and by using the retrieved user ID we query the edge collection recommend where the edge is bound to the user ID and edge description is of type property.



```

[
  {
    "_key": "504359",
    "_id": "recommend/504359",
    "_from": "User/1187045",
    "_to": "User/1187044",
    "_rev": "_YJO6cIi--P",
    "recommended_property": "property/69123:14"
  },
  {
    "_key": "504351",

```

```

    "_id": "recommend/504351",
    "_from": "User/1187042",
    "_to": "User/1187044",
    "_rev": "_YJO6cIi--",
    "recommended_property": "property/69123:8"
  },
  {
    "_key": "504361",
    "_id": "recommend/504361",
    "_from": "User/1187041",
    "_to": "User/1187044",
    "_rev": "_YJO6cIi--T",
    "recommended_property": "property/69123:4"
  }
]

```

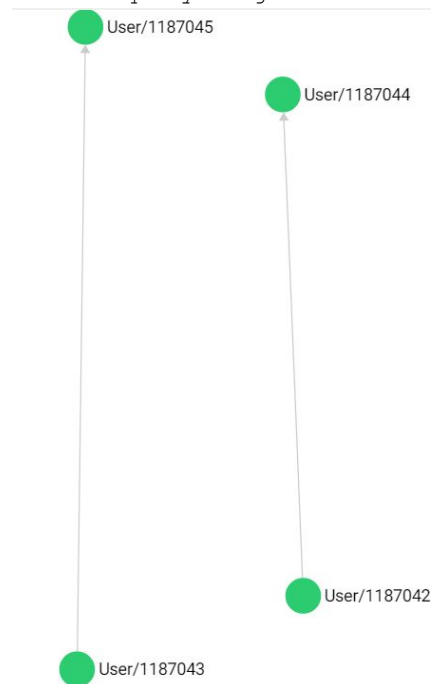
4.2 get to who all the property is recommended

```

for r in recommend
filter  r.recommended_property == "property/69123:8"
return r

```

Here we query Edge collection recommend where is Edge description is the given property ID.



```

[
  {
    "_key": "504351",
    "_id": "recommend/504351",
    "_from": "User/1187042",
    "_to": "User/1187044",
    "_rev": "_YJO6cIi--",
    "recommended_property": "property/69123:8"
  },
  {
    "_key": "504353",
    "_id": "recommend/504353",
    "_from": "User/1187043",
    "_to": "User/1187045",
    "_rev": "_YJO6cIi--D",
    "recommended_property": "property/69123:8"
  }
]

```

5.1 Updating priority weight dynamically.

```

let no_of_user = FIRST(for u in User
                        collect with count into cnt
                        return cnt)

```

```

for p in property

```



```
filter CONTAINS(p._key,"69123")

let nos = LENGTH(
  for v in 1..1 inbound CONCAT("property/",p._key)  shortlist
return 1)

let nov = LENGTH(p.viewers_user_id)

let slot = p.sort_priority.slot*100

let inter = ((nos+nov)/no_of_user)*100

update p with {"sort_priority":{"weight" : inter+slot }} into property

let w = inter+slot
sort w DESC

return { p : p._key , weight : w }
```

The property we have attribute slot under the section sort priority .the value of this slot for each and every property is 0 by default.Indonesian visas only 3 slots will be available. therefore for a region out of all properties any 3 properties can have the slot value as one two or three.We need total number of users ,number of subscribers and number of viewers for a particular property for calculating to sort weight.

For number of users we query the collection user and Aggregate using the with count clause.

For number of subscribers , we retrieve all the inbound edges to the given property ID in the edge collection shortlist.return one for each edge. then we get the count of this array which is nothing but the number of subscribers for that particular property ID.

We are storing the viewers of property under the tag viewers_user_id. to get the number of viewers we just get the length of this viewer array.

sort priority weight can be divided in 2 sections. First the natural way by combining the number of subscribers and number of viewers. second by the slot weight.

the natural weight is calculated as follows $((nos+nov)/no_of_user)*100$

The slot weight is calculated as follows $p.sort_priority.slot*100$

By adding these two values we get the total sort priority weight of a property.

Query

17 elements

45.426 ms

17 writes

0 writes ignored

p	weight
69123:3	400
69123:2	320
69123:1	260
69123:10	140
69123:5	120
69123:6	120
69123:4	80
69123:7	80
69123:8	80
69123:13	80
69123:15	60

```
[
  {
    "p": "69123:3",
    "weight": 400
  },
  {
    "p": "69123:2",
    "weight": 320
  },
  {
    "p": "69123:1",
    "weight": 260
  },
  {
    "p": "69123:10",
    "weight": 140
  },
  {
    "p": "69123:5",
    "weight": 120
  },
  {
    "p": "69123:6",
    "weight": 120
  },
  {
    "p": "69123:4",
    "weight": 80
  },
  {
    "p": "69123:7",
    "weight": 80
  },
  {
    "p": "69123:8",
    "weight": 80
  },
  {
    "p": "69123:13",
    "weight": 80
  }
]
```

6. Property in Demand

```
for p in property
filter left(p._key , 6) == "69123:"
let nov = LENGTH(p.viewers_user_id)

let nos =  LENGTH(
    for v in 1..1 inbound CONCAT("property/",p._key)  shortlist
    return 1)

SORT nov desc
SORT nos desc

return {id : p._key, "number of views" :nov,"number of shortlist" : nos }
```

The demand of the property can be decided by its number of viewers and number of shortlist. Here we use the similar logic to calculate these numbers as used in priority weight calculations.

Query 3 elements 1.836 ms

id	number of views	number of shortlist
69123:10	3	4
69123:1	5	3
69123:5	3	3

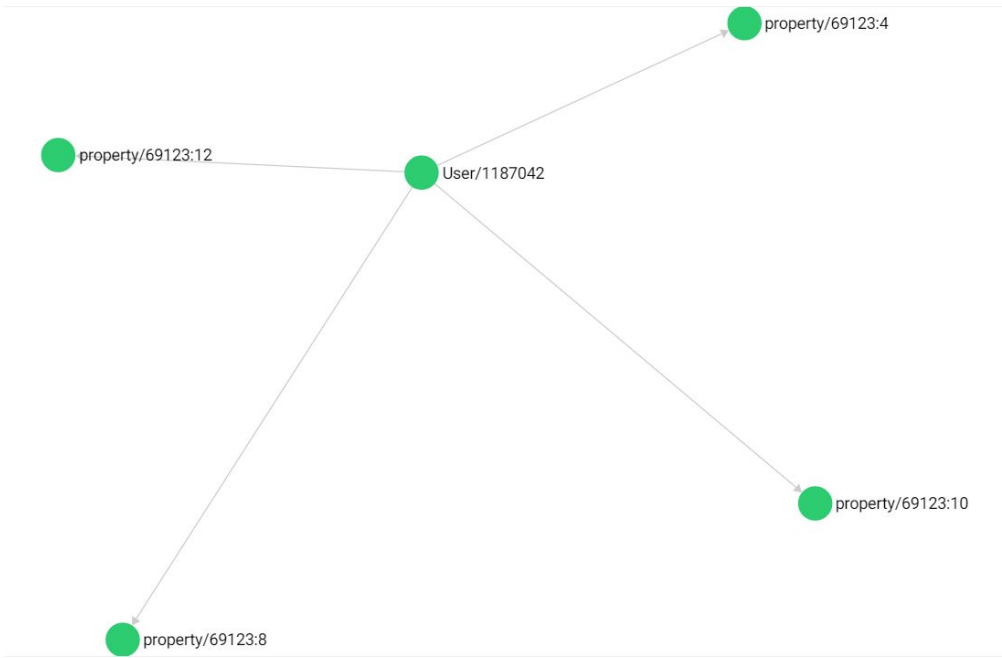
```
[
  {
    "id": "69123:10",
    "number of views": 3,
    "number of shortlist": 4
  },
  {
    "id": "69123:1",
    "number of views": 5,
    "number of shortlist": 3
  },
  {
    "id": "69123:5",
    "number of views": 3,
    "number of shortlist": 3
  }
]
```

7.1 Property shortlist by a user

```
let user_id = FIRST(for u in User filter u.user_info.first_name == "sanket" return u._id)
for v,e,p IN 1..1 outbound user_id shortlist
return e
```

He would be utilised the fact that we have a relation from user ID to a property and a description as shortlist.

Here me find all the edges outbound from a user in edge collection shortlist such that the max hops between them is of just one edge.



```
[
  {
    "_key": "1188429",
    "_id": "shortlist/1188429",
    "_from": "User/1187042",
    "_to": "property/69123:4",
    "_rev": "_YI9aZpa--",
    "type": "shortlist"
  },
  {
    "_key": "1188431",
    "_id": "shortlist/1188431",
    "_from": "User/1187042",
    "_to": "property/69123:8",
    "_rev": "_YI9aZrG--B",
    "type": "shortlist"
  },
  {
    "_key": "1188438",
    "_id": "shortlist/1188438",
    "_from": "User/1187042",
    "_to": "property/69123:10",
    "_rev": "_YJOxqJa--",
    "type": "shortlist"
  },
  {
    "_key": "1188439",
    "_id": "shortlist/1188439",
    "_from": "User/1187042",
    "_to": "property/69123:12",
    "_rev": "_YJOxqJa--B",
    "type": "shortlist"
  }
]
```



7.2 compare shortlisted properties

```
let user_id = FIRST(for u in User filter u.user_info.first_name == "sanket" return u._id)

let s_p = (for s in shortlist filter s._from == user_id return s._to)

for s in s_p
  for p in property
    filter p._key == REGEX_REPLACE(s,"property/", "")
    return { key : p._key, Price : p.amount.base_price , size : p.advertise.property_title }
```

once the user has the shortlisted properties you might like to compare them. Using the user ID and shortlisted properties as we did in section 7.1 we can use it for the to query the collection property and the desired attributes to compare.

Query  4 elements  1.457 ms 		
key	Price	size
69123:4	1400	1 bhk
69123:8	287	3.5 bhk
69123:10	500	3.5 bhk
69123:12	950	3.5 bhk

```
[
  {
    "key": "69123:4",
    "Price": 1400,
    "size": "1 bhk"
  },
  {
    "key": "69123:8",
    "Price": 287,
    "size": "3.5 bhk"
  }
]
```

```

},
{
  "key": "69123:10",
  "Price": 500,
  "size": "3.5 bhk"
},
{
  "key": "69123:12",
  "Price": 950,
  "size": "3.5 bhk"
}
]

```

8.1 get all the user subscribed to get notified for a new property in 69123

```

for r in notification
filter  r._to == "property/69123"
return r

```

For maintaining the notification squad of users we maintain a relation from the user to the PIN code in the edge collection notification. utilising this fact, to get all the users subscribed we filter edges with _to as PIN code and retrieve all the notification edges.



```

[
  {
    "_key": "1168399",
    "_id": "notification/1168399",
    "_from": "User/1187042",
    "_to": "property/69123",
    "_rev": "_YI9brCi--",
    "type": "subscription"
  },
  {
    "_key": "1168412",
    "_id": "notification/1168412",
    "_from": "User/1187045",
    "_to": "property/69123",
    "_rev": "_YI9brCi--B",
    "type": "subscription"
  },
  {
    "_key": "1168523",
    "_id": "notification/1168523",
    "_from": "User/1187043",
    "_to": "property/69123",

```

```

    "_rev": "_YI9brE---",
    "type": "subscription"
  },
  {
    "_key": "1168537",
    "_id": "notification/1168537",
    "_from": "User/1187044",
    "_to": "property/69123",
    "_rev": "_YI9brE---B",
    "type": "subscription"
  }
]

```

8.2 get all the users who were notified when 69123:5 was listed

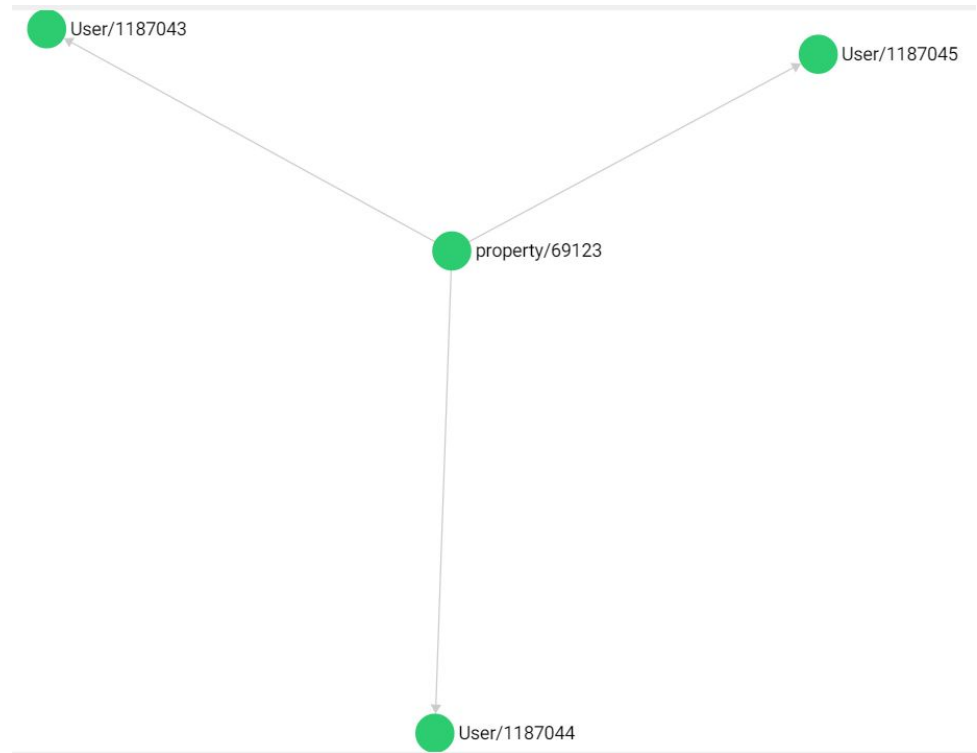
```

for r in notification
filter r.notification_of == "property/69123:5"
return r

```

Whenever a new properties added to a particular PIN code the users subscribe to it must be notified about this new property. we take care of this consequential insert as shown in section 1.1. the edge description of such cases would be from a PIN code to a user with description as the new property ID.

from this we can get all the users that were notified when a particular new property is listed by querying over the edge description as property id.



```

[
  {
    "_key": "1169483",
    "_id": "notification/1169483",
    "_from": "property/69123",
    "_to": "User/1187045",
    "_rev": "_YI9brCi--D",
    "notification_of": "property/69123:5"
  },
  {
    "_key": "1168538",
    "_id": "notification/1168538",
    "_from": "property/69123",
    "_to": "User/1187044",
    "_rev": "_YI9brE---F",
    "notification_of": "property/69123:5"
  },
  {
    "_key": "1168524",

```

```

    "_id": "notification/1168524",
    "_from": "property/69123",
    "_to": "User/1187043",
    "_rev": "_YI9brEC--",
    "notification_of": "property/69123:5"
  }
]

```

9.1 search nearby locations

```

for p in property
  filter p._key == "69123:2"
  for l in property
    filter CONTAINS(l._key,'super_market')
    let d = DISTANCE(p.coordinate[0], p.coordinate[1],l.coordinate[0], l.coordinate[1])
    SORT d
    limit 3
    return {near_by : l._key ,"Aerial distance(km)" : d/1000}

```

Along with the property attribute we also store its geo coordinates locations. Also we can create the geindex on these coordinates. this opens up the possibilities of using the geo functions made available in arangoDB.

in this query we aim to find the nearby supermarkets from the property Id 2. by using the distance function we can calculate the distance between the coordinates of property ID 2 and all the supermarket property coordinates. Later by sorting them in ascending order and limiting them to 3 we can get the nearby supermarkets and it's Aerial distance from the given property ID.

near_by	Aerial distance(km)
super_market:4	4.245537934023409
super_market:2	5.284532326265886
super_market:1	6.083444080457969

```

[
  {
    "near_by": "super_market:4",
    "Aerial distance(km)": 4.245537934023409
  },
  {
    "near_by": "super_market:2",
    "Aerial distance(km)": 5.284532326265886
  },
  {
    "near_by": "super_market:1",
    "Aerial distance(km)": 6.083444080457969
  }
]

```

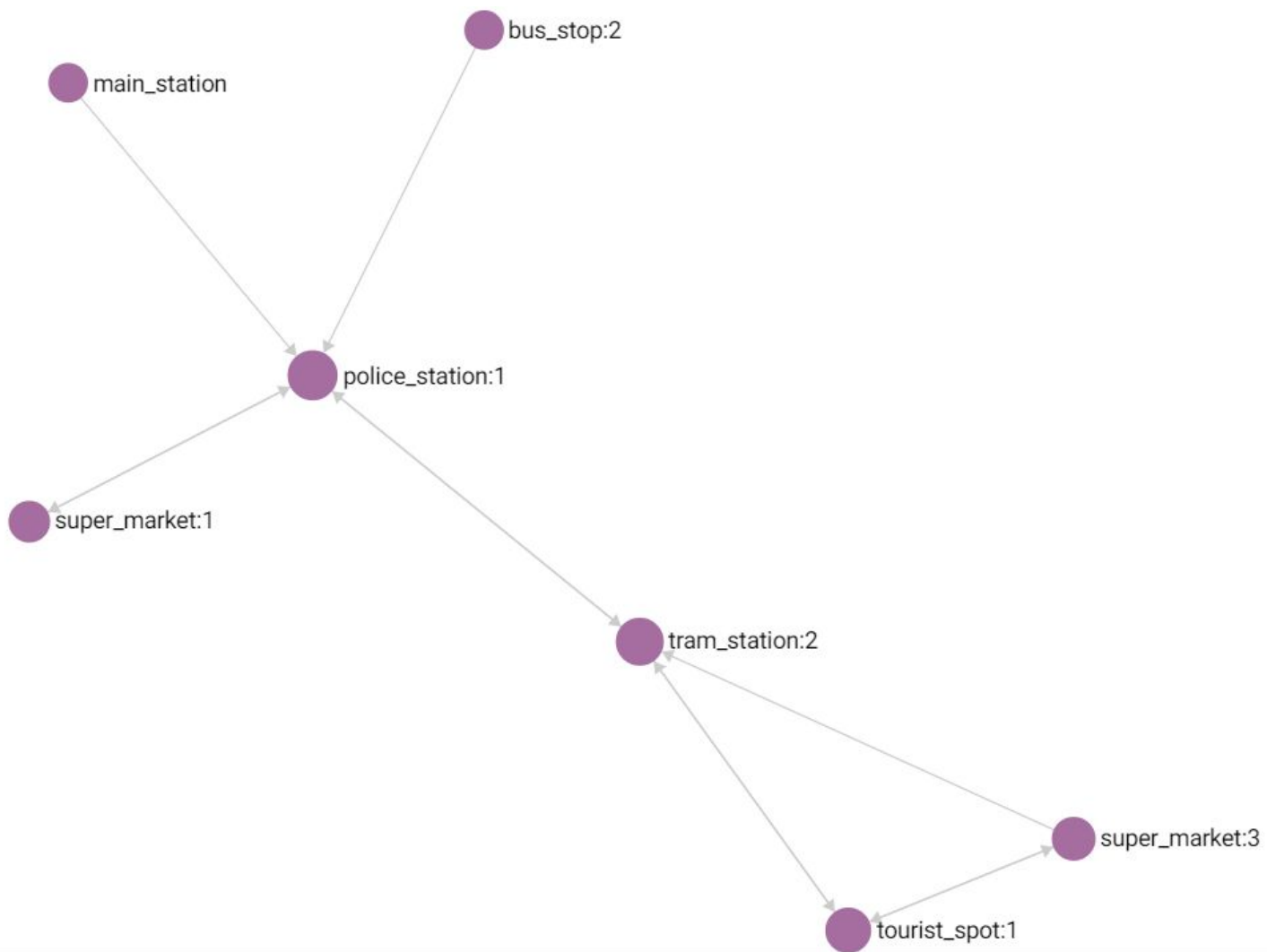
9.2 shortest path in number of hops

```

FOR v,e IN OUTBOUND SHORTEST_PATH
  "property/69123:1" TO "property/bus_stop:1"
  map
  return {from : e._from , to : e._to, distance : e.distance}

```

For creating a virtual map between the properties and locational amenities we created a collection which includes the edge from a property to its two near most properties with edge description as distance between them..



Note that the distance between the Edges is the aerial distance and not the actual path distance. to get the actual path distance we need to extend this implementation by introducing the path nodes from these properties.

Once we have established the edges between the nodes we have various traversal syntaxes in arango. the special syntax for finding the shortest path between two edges. in this we have to specify the start node and end node between which we have to find the shortest path. by using the syntax we can fetch all the information regarding the vertices and edges we come across the path.

Query 3 elements 0.773 ms		
from	to	distance
property/69123:1	property/69123:3	3.604656579993115
property/69123:3	property/69123:6	0.7272123482966897
property/69123:6	property/bus_stop:1	0.18247723594862222

```

[
  {
    "from": "property/69123:1",
    "to": "property/69123:3",
    "distance": 3.604656579993115
  },
  {
    "from": "property/69123:3",
    "to": "property/69123:6",
    "distance": 0.7272123482966897
  },
  {
    "from": "property/69123:6",
    "to": "property/bus_stop:1",
  }
]

```



```
    "distance": 0.18247723594862222  
  }  
]
```