FIT3176 Assignment 1: MongoDB & Cassandra

Group 29
Darren Luwi (29051754)
Scott Hicks (31556140)

CONTENTS:

Contribution Declaration Form

(to be completed by all team members)

Please fill in the form with the contribution from each student towards the assignment.

1 NAME AND CONTRIBUTION DETAILS

Student ID	Student Name	Contribution Percentage
31556140	Scott Hicks	50
29051754	Darren Luwi	50

2 DECLARATION

We declare that:

- The information we have supplied in or with this form is complete and correct.
- We understand that the information we have provided in this form will be used for individual assessment of the assignment.

3 SIGNATURE

Signatures

Day Month Year

Date 2 / 9 / 2022

Group Assignment Cover Sheet

Student ID Number	Surname	Given Names
31556140	Hicks	Scott
29051754	Luwi	Darren

* Please include the names of all other group members.

Unit name and code	Advanced Databases - FIT3176	
Title of assignment	Assignment 1: MongoDB & Cassandra	
Lecturer/tutor	Farah Kabir	
Tutorial day and time	Friday - 4PM	**Campus** Clayton
Is this an authorised group assignment? ☐ **Yes** ☐ **No**		
Has any part of this assignment been previously submitted as part of another unit/course? ☐ **Yes** ☐ **No**		
Due Date 16/9/22 11:55 PM		Date submitted 16/9/22

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 The material used can be from any source (staff, students or the internet, published and unpublished works).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
- I have taken proper care to safeguard this work and made all reasonable efforts to ensure it could not be copied.
- No part of this assignment has been previously submitted as part of another unit/course.
- I acknowledge and agree that the assessor of this assignment may for the purposes of assessment, reproduce the assignment and:
  - i. provide to another member of faculty and any external marker; and/or
  - ii. submit it to a text matching software; and/or
  - iii. submit it to a text matching software which may then retain a copy of the assignment on its database for the purpose of future plagiarism checking.
- I certify that I have not plagiarised the work of others or participated in unauthorised collaboration when preparing this assignment.

Signature ..... *Date: 16/9/22*

* delete (iii) if not applicable

Signature _____ Date: 16/9/22

Signature _____ _____ Date: _____ 16/9/22 _____

Privacy Statement
The information on this form is collected for the primary purpose of assessing your assignment and ensuring the academic integrity requirements of the University are met. Other purposes of collection include recording your plagiarism and collusion declaration, attending to course and administrative matters and statistical analyses. If you choose not to complete all the questions on this form it may not be possible for Monash University to assess your assignment. You have a right to access personal information that Monash University holds about you, subject to any exceptions in relevant legislation. If you wish to seek access to your personal information or inquire about the handling of your personal information, please contact the University Privacy Officer: privacyofficer@adm.monash.edu.au

FIT3176 Assignment 1: MongoDB & Cassandra

A. General Information and Submission

o This is a group assignment. One group consists of **2 students**.
o *Submission method*: Submission is online through Moodle.
o *Penalty for late submission*: 10% deduction for each day (including weekends). o *Assignment Cover Sheet*: You will need to sign the assignment cover sheet. o *Contribution Form*: The contribution needs to be completed by all members and Please sign (e-signature is acceptable) the form as an agreement between members. o Please carefully read the requirements for EACH section, especially the Task Outputs.

B. Problem Description – MonUGov

MonUGov is a Monash University initiative which utilizes open source government data to provide services to the Monash community. One service MonUGov provides is helping individuals find housing options according to their preferences in budget, property type, area, location etc. They do this by accessing various datasets given in the vic.gov.au site and doing analysis to match the individual's preferences.

MonUGov has hired your team of Advanced Database Experts to use the following sample data files downloaded from vic.gov.au to help with the data analysis that helps MonUGov provide their services:

- suburbs.csv
- landmarks.csv
- properties.json
- properties.csv

Note: These data are raw data that does not follow any particular schema. For the

analysis MonUGov has asked your team to perform the following tasks:

C. Tasks

The assignment is divided into **FOUR** main tasks:

Since MonUGov has heard about both `MongoDB` and `Cassandra`, therefore, they wish to use a combination of both technologies to analyse the data.

C.1. Analysis using `MongoDB`.

***Data Requirement*:**	The data for this task is contained in the following files: • `suburbs.csv` • `landmarks.csv` • `properties.json`
Software Requirement:	(i) `MongoDB Compass`. (ii) A software that can run Mongo Shell commands (e.g. `PyMongo, Terminal, Command Prompt` etc.).
Overall Task C.1. Outputs:	(i) **Screenshots** added to the reports for each question as detailed in each question, with before and after screenshots for any `insert/update/delete` made to the database. (ii) All **code** added in a properly **formatted and commented file** named **`C1_MongoDB.js`**

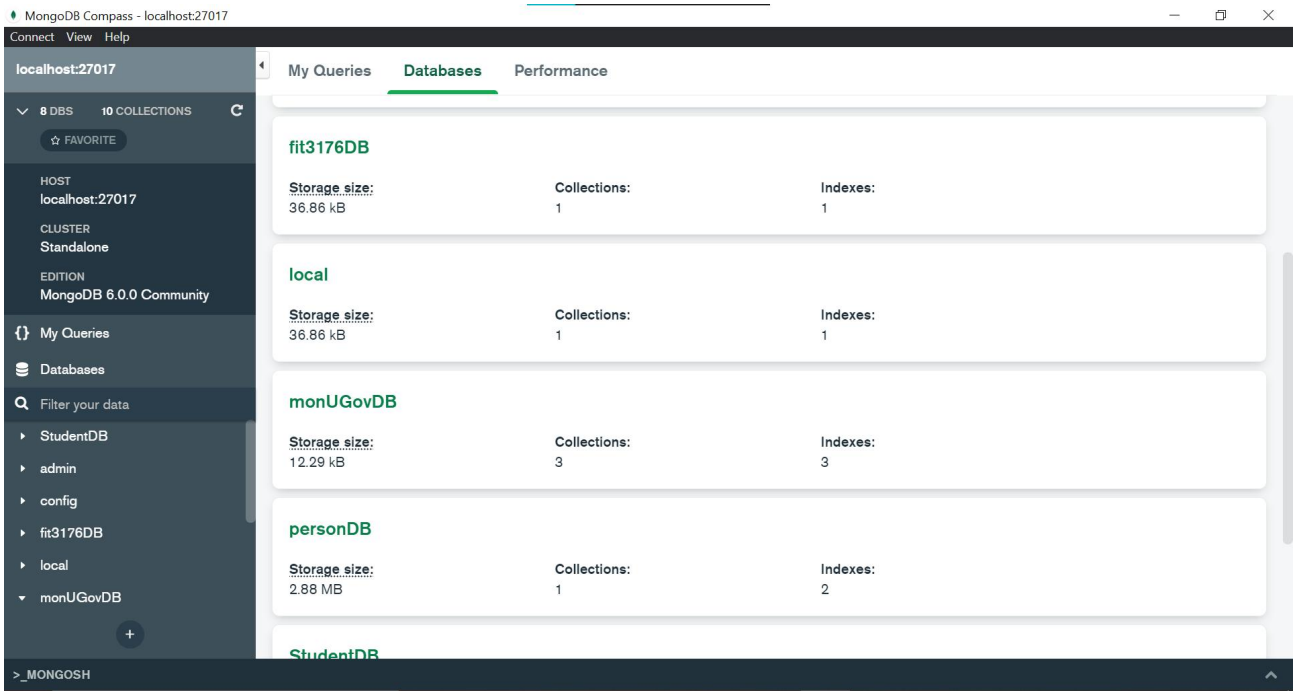
Task Requirements:

The following tasks require the use of `MongoDB Compass`:

C.1.1. Create a database called **`monUGovDB`**.

Provide in your report a screenshot of the created database in the **list of all databases**.

SCREENSHOT:



C.1.2. In the newly created database using appropriate data types add the data from a. `suburbs.csv` into the `suburbs` collection

b. `landmarks.csv` into the `landmarks` collection

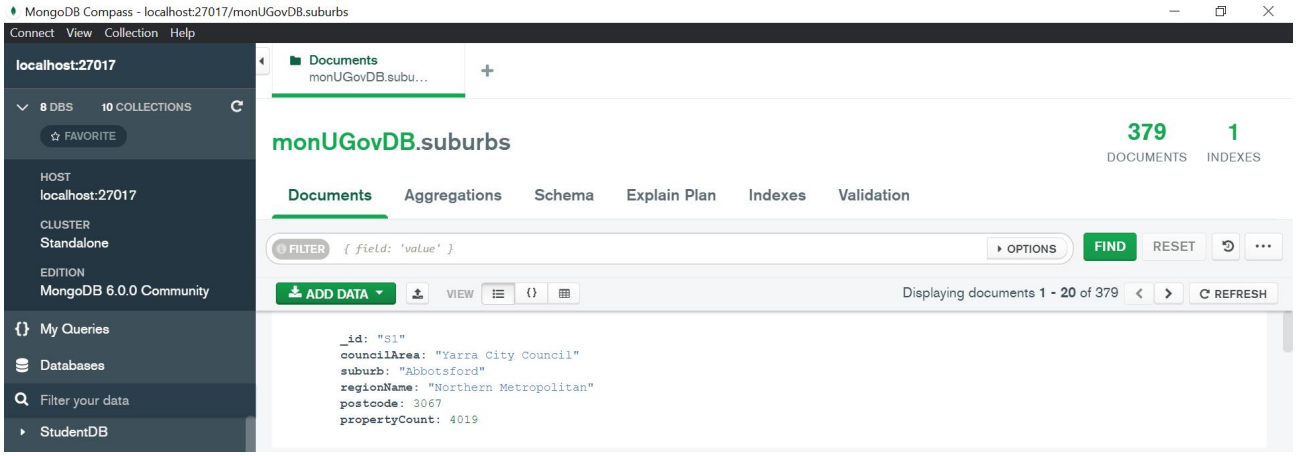
c. `properties.json` into the `properties` collection

Provide in your report a screenshot of **one document** in each created collection after adding the data.

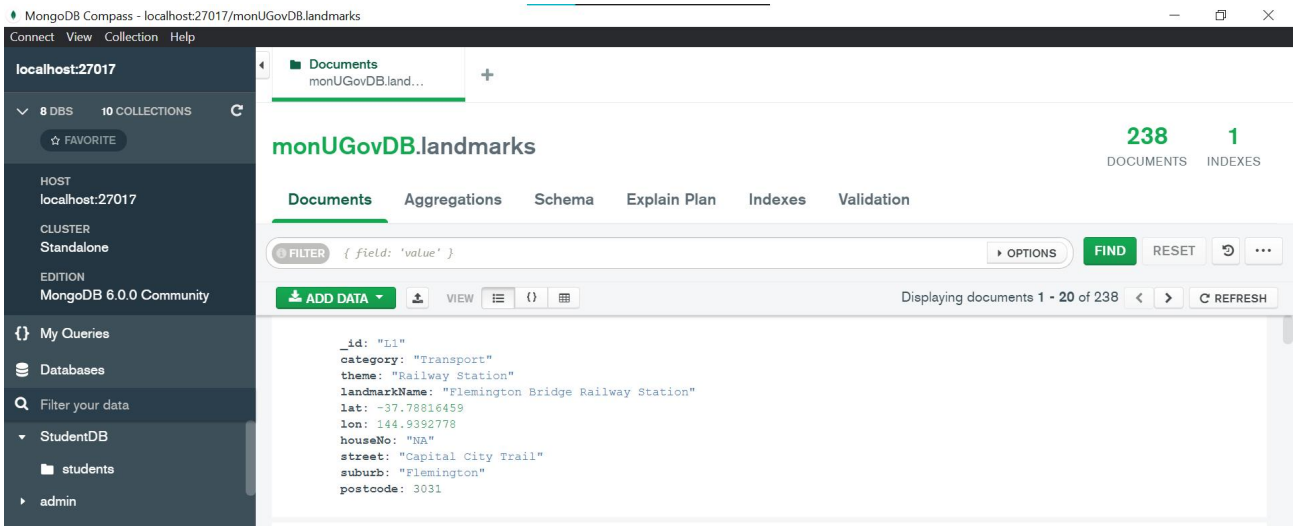
Note: Please check each field and the queries from section C1.4 to assign the relevant data types either while or after importing. *More than one collection can be used to save any modified documents; however the result of C1.2 should be the 3 collections mentioned above with correct names:* `suburbs`, `landmarks` *and* `properties`.

SCREENSHOTS:

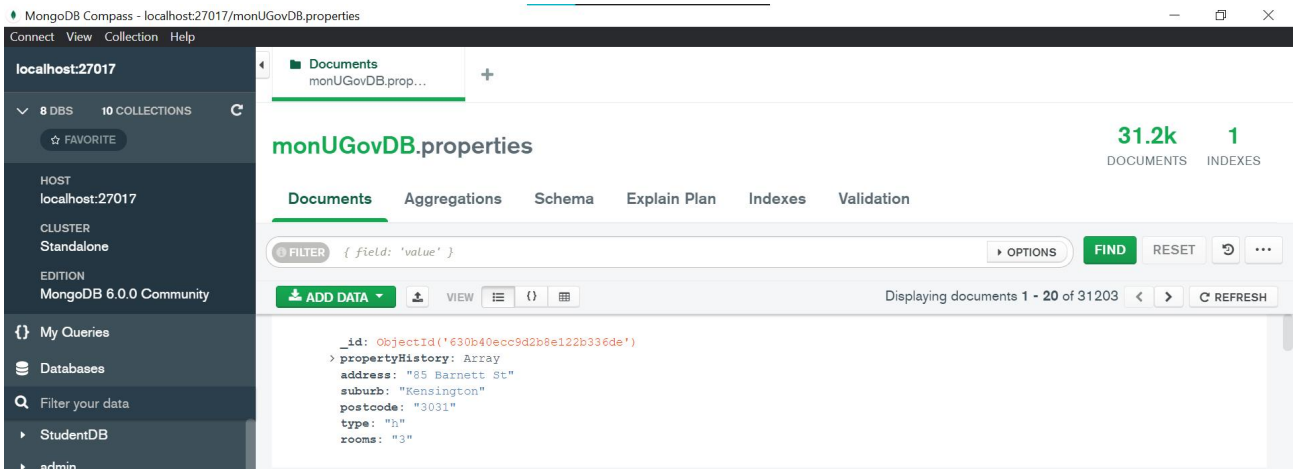
Suburbs Collection:



Landmarks Collection:



Properties Collection:



The following tasks require the use of `MongoDB Shell`. Where applicable and unless stated otherwise you can use either MongoDB CRUD methods, single-purpose aggregations, or aggregation pipeline to answer the tasks.

Note: *Marks for this section depend on the* ***query efficiency*** *e.g. the processing speed, number of documents scanned, the storage, the number of queries used, etc. Therefore, using more than the required amount of queries to answer a section or using temporary variables, collections, cursors (e.g. for each loop) may incur mark penalties.*

C.1.3. Read the queries from C1.4 and create one single field index and one compound (more than one field) index to help speed up at least one query.
Provide in your report the code to create the indices and screenshots of the created indices details.

Ans:
The "text" Single Field Index code will be useful for C.1.4.1, as it speeds up and eases the query operation, which involves searching for a specific text/string.

Meanwhile, the "Suburb" and "Address" Compound Index Code will come in handy for counting the number of properties in C.1.4.2 as every property has different addresses, while almost every question in section C.1.4 involves suburbs.

CODE:

Single Field Index Code:

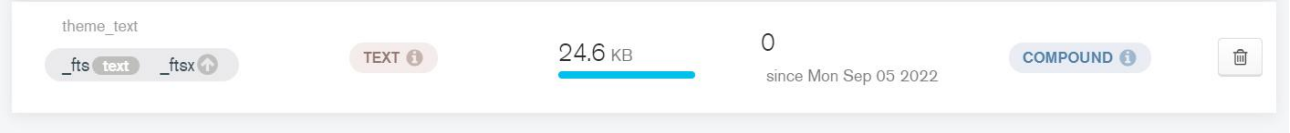
```
db.landmarks.createIndex({theme:"text"})
```

Compound Index Code:

```
db.properties.createIndex({"suburb":1, "address":1})
db.suburbs.createIndex({"suburb":1, "address":1})
```

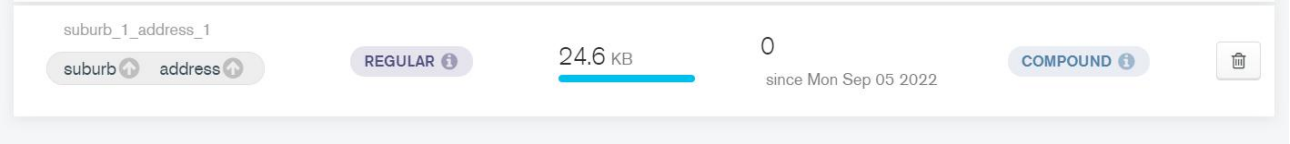
SCREENSHOTS:

Landmarks:



Properties:

Suburbs:



C.1.4. The following questions are MongoShell queries:

(i) List the **landmarks** that had a theme of "School", for example, but not limited to "Secondary Schools", "Primary Schools", "School - Primary and Secondary Education" etc.
Provide the MongoShell query code and a screenshot of the MongoDB Shell output containing the `landmarkName` and theme name in the following format:

```
"landmarkName": _________,
"theme": _________
```

CODE:

```
db.landmarks.find({$text:{$search: "school"}}, {landmarkName:1,
theme:1, _id:0})
```

SCREENSHOT:

```
> db.landmarks.find({$text:{$search: "school"}}, {landmarkName:1, theme:1, _id:0})
< { theme: 'Primary Schools',
    landmarkName: 'North Melbourne Primary School' }
  { theme: 'Primary Schools',
    landmarkName: 'Kensington Primary School' }
  { theme: 'Secondary Schools',
    landmarkName: 'University High School' }
  { theme: 'Secondary Schools',
    landmarkName: 'Melbourne Grammar School' }
  { theme: 'Primary Schools',
    landmarkName: 'Carlton Primary School' }
  { theme: 'Primary Schools',
    landmarkName: 'Carlton Gardens Primary School' }
  { theme: 'School - Primary and Secondary Education',
    landmarkName: 'Melbourne Girls Grammar School' }
  { theme: 'School - Primary and Secondary Education',
    landmarkName: 'Wesley College' }
```

(ii) Count the number of **properties** in each suburb and list all suburbs from highest to lowest property count.
Provide the MongoShell query code and a screenshot of the MongoDB Shell output containing the `suburb` name and the number of properties as `propertyCount` in the following format:

```
"suburb": _________,
"propertyCount": _________
```

CODE:

```
db.suburbs.find({},{"suburb":1, "propertyCount":1, _id:0}).sort({"propertyCount":-1})
```

SCREENSHOT:

```
> db.suburbs.find({},{"suburb":1, "propertyCount":1, _id:0}).sort({"propertyCount":-1})
< { suburb: 'Reservoir', propertyCount: 21650 }
  { suburb: 'Melbourne', propertyCount: 17496 }
  { suburb: 'Pakenham', propertyCount: 17384 }
  { suburb: 'Berwick', propertyCount: 17093 }
  { suburb: 'Frankston', propertyCount: 17055 }
  { suburb: 'Werribee', propertyCount: 16166 }
  { suburb: 'Point Cook', propertyCount: 15542 }
  { suburb: 'Craigieburn', propertyCount: 15510 }
  { suburb: 'Glen Waverley', propertyCount: 15321 }
  { suburb: 'Richmond', propertyCount: 14949 }
  { suburb: 'South Yarra', propertyCount: 14887 }
  { suburb: 'Preston', propertyCount: 14577 }
  { suburb: 'Sunbury', propertyCount: 14092 }
  { suburb: 'St Albans', propertyCount: 14042 }
  { suburb: 'Hoppers Crossing', propertyCount: 13830 }
  { suburb: 'Mount Waverley', propertyCount: 13366 }
  { suburb: 'St Kilda', propertyCount: 13240 }
  { suburb: 'Croydon', propertyCount: 11925 }
  { suburb: 'croydon', propertyCount: 11925 }
  { suburb: 'Brunswick', propertyCount: 11918 }
  Type "it" for more
```

(iii) Using MongoDB Aggregation Pipeline display the **Council Area** having the **second highest average property count**. Display the result in the following format with `avgPropertyCount` rounded to **1 decimal place**:

```
"councilArea": _________,
"avgPropertyCount": _________
```

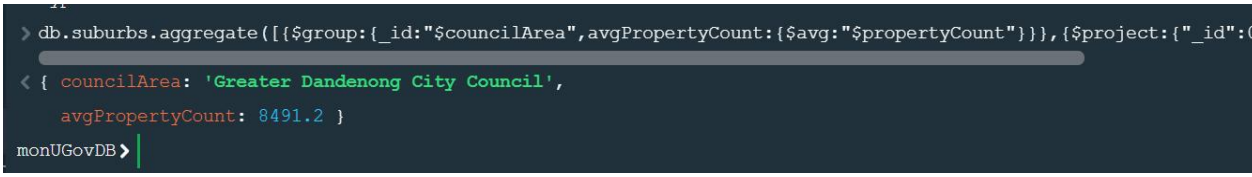
Provide the MongoShell query code and a screenshot of the MongoDB Shell output containing the `councilArea` name of the council and the `avgPropertyCount` fields.

CODE:

```
db.suburbs.aggregate([{$group:{_id:"$councilArea",avgPropertyCount:{$a
vg:"$propertyCount"}}},{$project:{"_id":0,"councilArea":"$_id",avgProp
```

```
ertyCount:{$round:["$avgPropertyCount",1]}}},{$sort:{"avgPropertyCount
":-1}},{$skip:1},{$limit:1}])
```

SCREENSHOT:



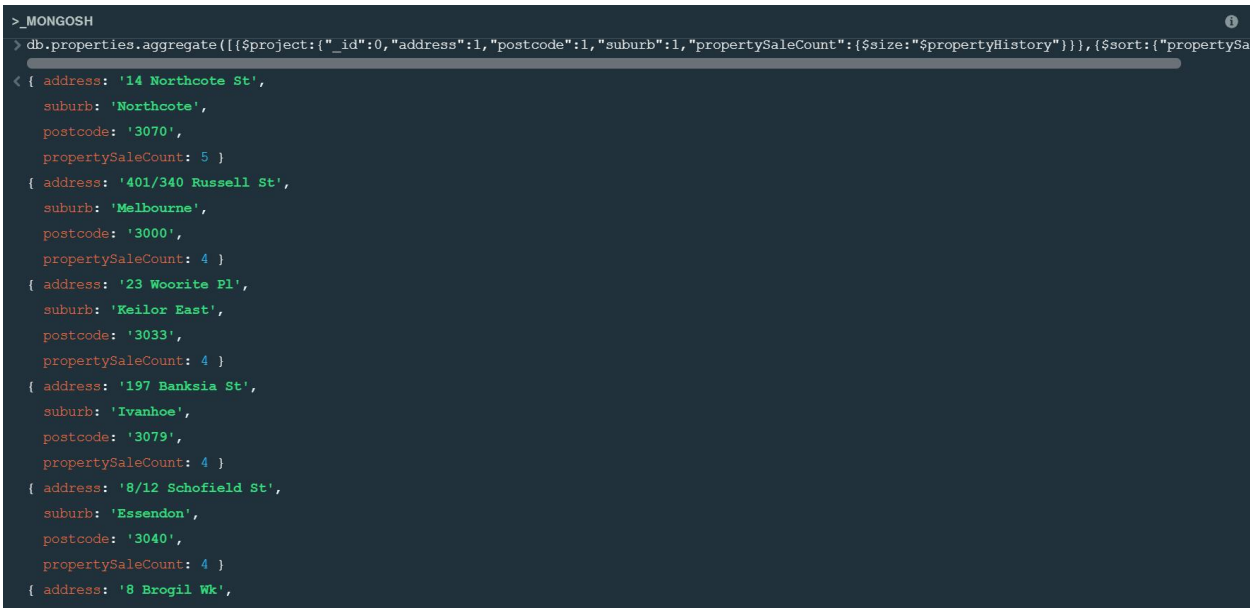
(iv) List the address, suburb, and postcodes of all **properties** sold more than once ordered from highest to lowest number of times the property was sold.
Provide the MongoShell query code and a screenshot of the MongoDB Shell output containing the property's `address`, `suburb`, `postcode`, and number of times it was sold as `propertySaleCount` in the following format:

```
"address": _________,
"postcode": _________,
"suburb": _________,
"propertySaleCount": _________
```

CODE:

```
db.properties.aggregate([{$project:{"_id":0,"address":1,"postcode":1,"
suburb":1,"propertySaleCount":{$size:"$propertyHistory"}}},{$sort:{"pr
opertySaleCount":-1}}])
```

SCREENSHOT:



C.1.5. Add a new field in the properties collection to contain new `type` values reflecting the following data:

old type	new type
h	house
u	unit
t	town house

If the type of property is neither `"h"`, `"u"` or `"t"`, then use `"other"` as the new type and save the updated collection as a new collection with name `propertiesNewTypes`.

Provide the MongoShell code for the update and a screenshot before and after the update from one document of each type of property in the following format:

Output Before Update:

```
"address": "85 Barnett St",
"postcode": 3031,
"suburb": "Kensington",
"type": "h"
```

Output After Update:

```
"address": "85 Barnett St",
```

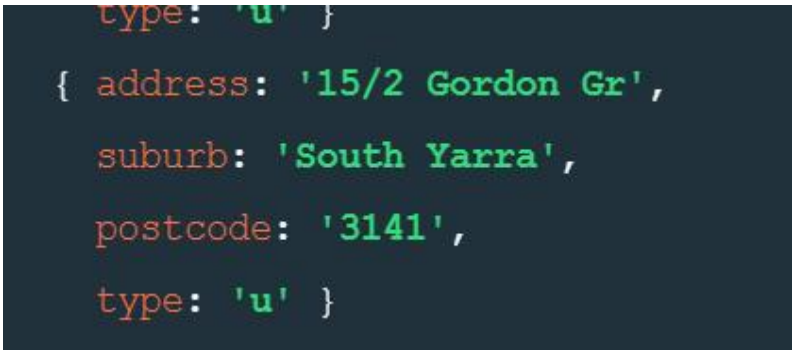
```
"postcode": 3031,
"suburb": "Kensington",
"type": "house"
```

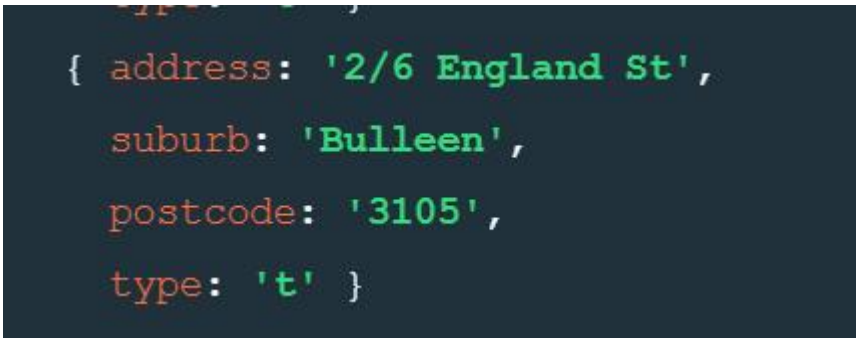
CODE:

```
db.properties.updateMany(
   { },
   [
     { $set: { type: { $switch: {
                            branches: [
                                { case: { $eq: [ "$type", "h" ] }, then:
"house" },
                                { case: { $eq: [ "$type", "u" ] }, then:
"unit" },
                                { case: { $eq: [ "$type", "t" ] }, then: "town
house" }
                            ],
                            default: "other"
     } } } }
   ]
)
```

SCREENSHOTS:

BEFORE UPDATE:

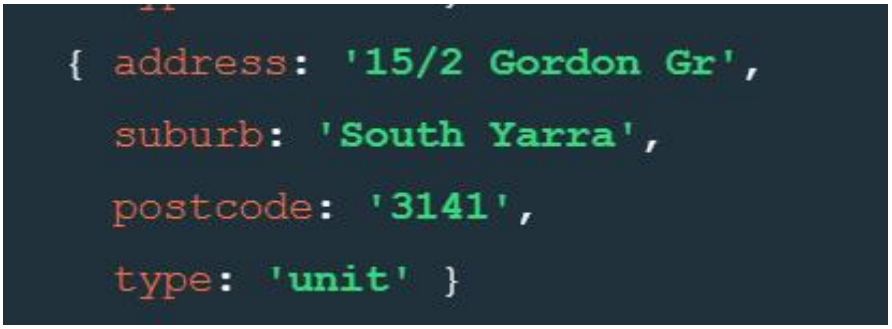




CODE SCREENSHOT:

```
> db.properties.updateMany(
    { },
    [
      { $set: { type: { $switch: {
                          branches: [
                              { case: { $eq: [ "$type", "h" ] }, then: "house" },
                              { case: { $eq: [ "$type", "u" ] }, then: "unit" },
                              { case: { $eq: [ "$type", "t" ] }, then: "town house" }
                          ],
                          default: "other"
      } } } }
    ]
  )
< { acknowledged: true,
    insertedId: null,
    matchedCount: 31203,
    modifiedCount: 31203,
    upsertedCount: 0 }
monUGovDB>
```

AFTER UPDATE:



```
{ address: '2/6 England St',
  suburb: 'Bulleen',
  postcode: '3105',
  type: 'town house' }
```

```
> db.properties.find({"type":"other"}, {"_id":0,"address":1,"postcode":1,"suburb":1, "type":1})
<
```

C.1.6. Using the `landmarks` collection find the documents that contain the `landmark` located on the `street` named `Monash Road` and update it to add a new field called `homeGround` with value `true` and a new field called `team` with value `yourGroupNo`.

Provide the MongoShell code for the update and a screenshot before and after the update.

Note: *the `yourGroupNo` will be different for each group attempting the FIT3176 assignment.*

CODE:

```
db.landmarks.update({"street": "Monash Road"},{$set: {"homeGround":true,
"team":29}})
```

SCREENSHOTS:

BEFORE UPDATE:

```
{ _id: 'L193',
  category: 'Education Centre',
  theme: 'Tertiary (University)',
  landmarkName: 'University of Melbourne',
  lat: Decimal128("-37.79828923"),
  lon: Decimal128("144.9609952"),
  houseNo: '163',
  street: 'Monash Road',
  suburb: 'Parkville',
  postcode: 3052 }
```

CODE SCREENSHOT:

```
> db.landmarks.update({"street": "Monash Road"},{$set: {"homeGround":true, "team":29}})
'DeprecationWarning: Collection.update() is deprecated. Use updateOne, updateMany, or bulkWrite.'
{ acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0 }
```

AFTER UPDATE:

```
{ _id: 'L193',
  category: 'Education Centre',
  theme: 'Tertiary (University)',
  landmarkName: 'University of Melbourne',
  lat: Decimal128("-37.79828923"),
  lon: Decimal128("144.9609952"),
  houseNo: '163',
  street: 'Monash Road',
  suburb: 'Parkville',
  postcode: 3052,
  homeGround: true,
  team: 29 }
```

C.1.7. This question is related to MongoDB joins:

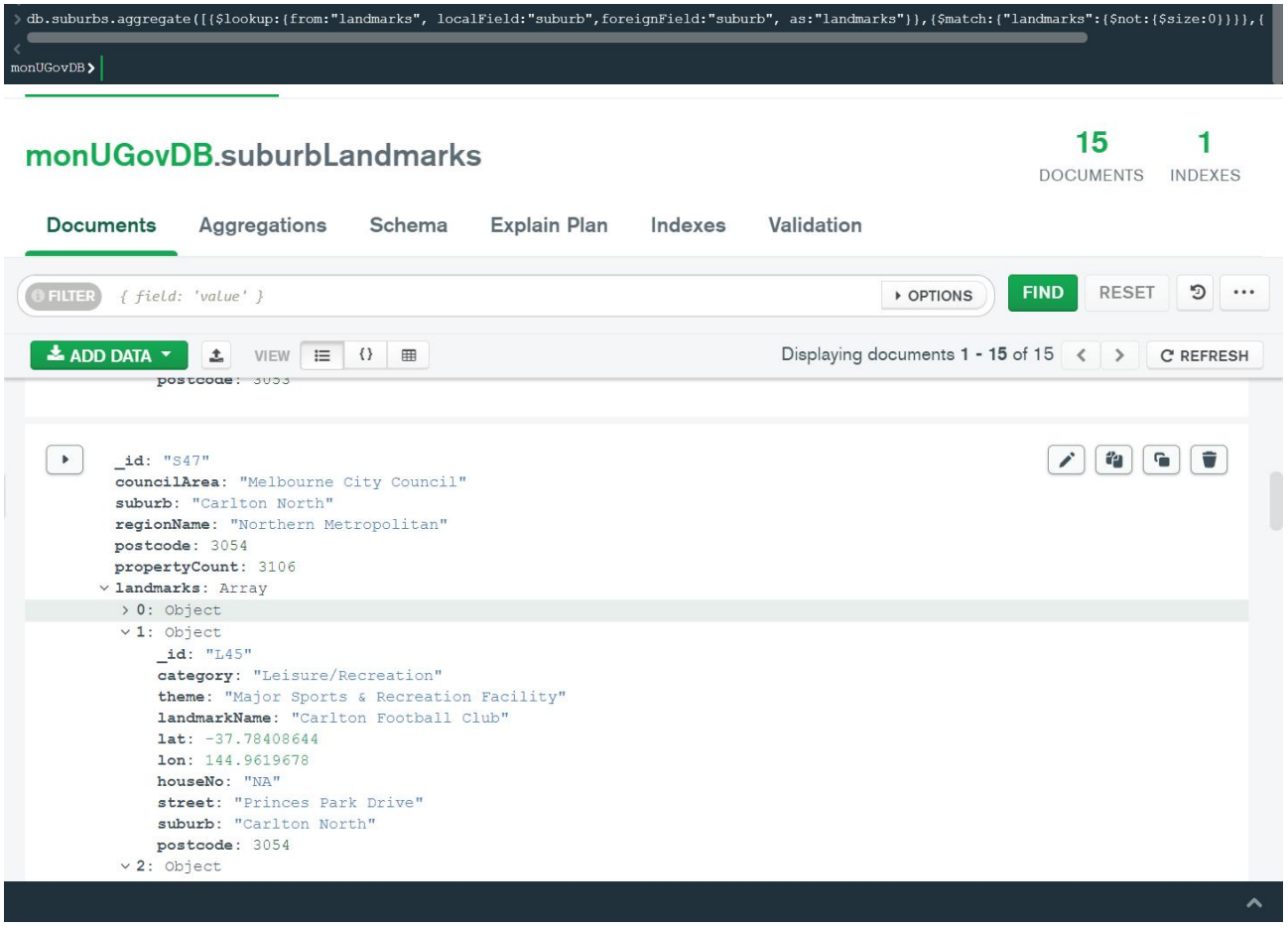
(i) Modify the documents in the `suburbs` collection so that the `landmarks` collection is embedded within the `suburbs` collection for documents that contain landmarks as an array field called `landmarks`. Store all suburbs containing a Landmark in a new collection called `suburbLandmarks`. Provide the MongoShell code and a screenshot of one document after the join.

Note: *You may want to double check while joining if the common field has the same value but different cases e.g. "abcd" and "Abcd" represent the same value*

CODE:

```
db.suburbs.aggregate([{$lookup:{from:"landmarks",
localField:"suburb",foreignField:"suburb",
as:"landmarks"}},{$match:{"landmarks":{$not:{$size:0}}}},{$out:"suburbLa
ndmarks"}])
```

SCREENSHOTS:



(ii) Display the region name with the maximum number of landmarks. Provide the MongoShell query code and a screenshot of the MongoDB Shell output containing the `regionName` and how many landmarks the region contained as `regionLandmarksCount` in the following format:
`"regionLandmarksCount": _____,`
`"regionName": "_____"`

CODE:

```
db.suburbLandmarks.aggregate([{$project:{"_id":0,"regionLandmarksCount":{$size:"$landmarks"},"regionName":1}},{$sort:{"regionLandmarksCount":-1}},{$limit:1}])
```

SCREENSHOTS:

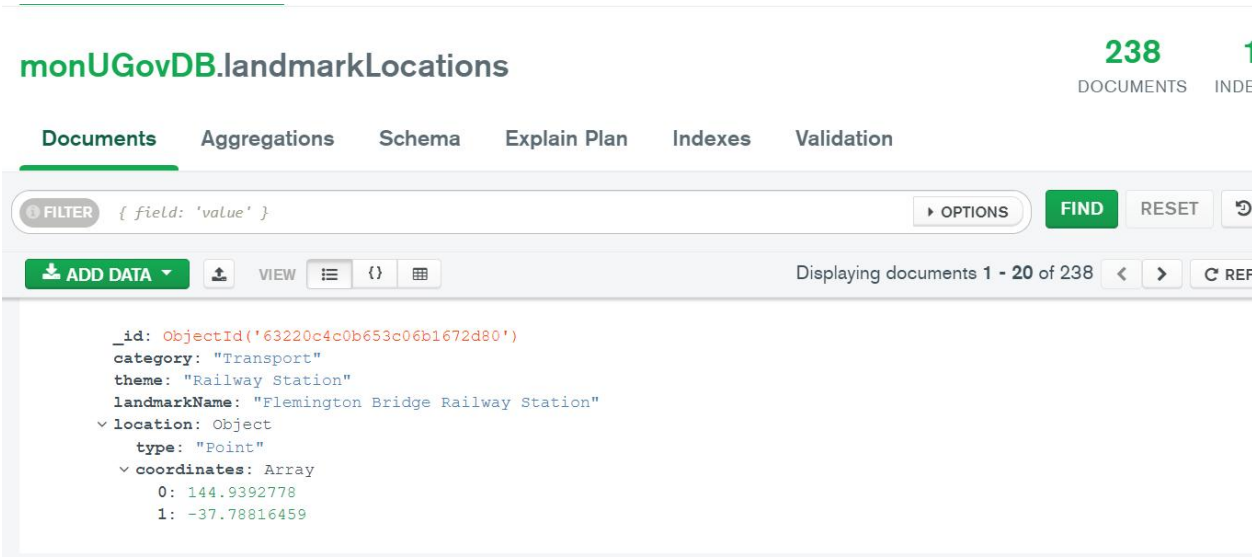
C.1.8. This question is related to MongoDB GeoSpatial queries:

(i) Using MongoDBShell commands modify the `landmarks` collections so that all `lat` and `lon` location coordinates are in MongoDB GeoJSON objects with location type as a `point` in a field called `location`.
Store the `landmarkName`, `category`, `theme` and `location` fields of the modified documents in a new collection called `landmarkLocations`.
Provide the MongoShell code and a screenshot of one document after the update.

CODE:

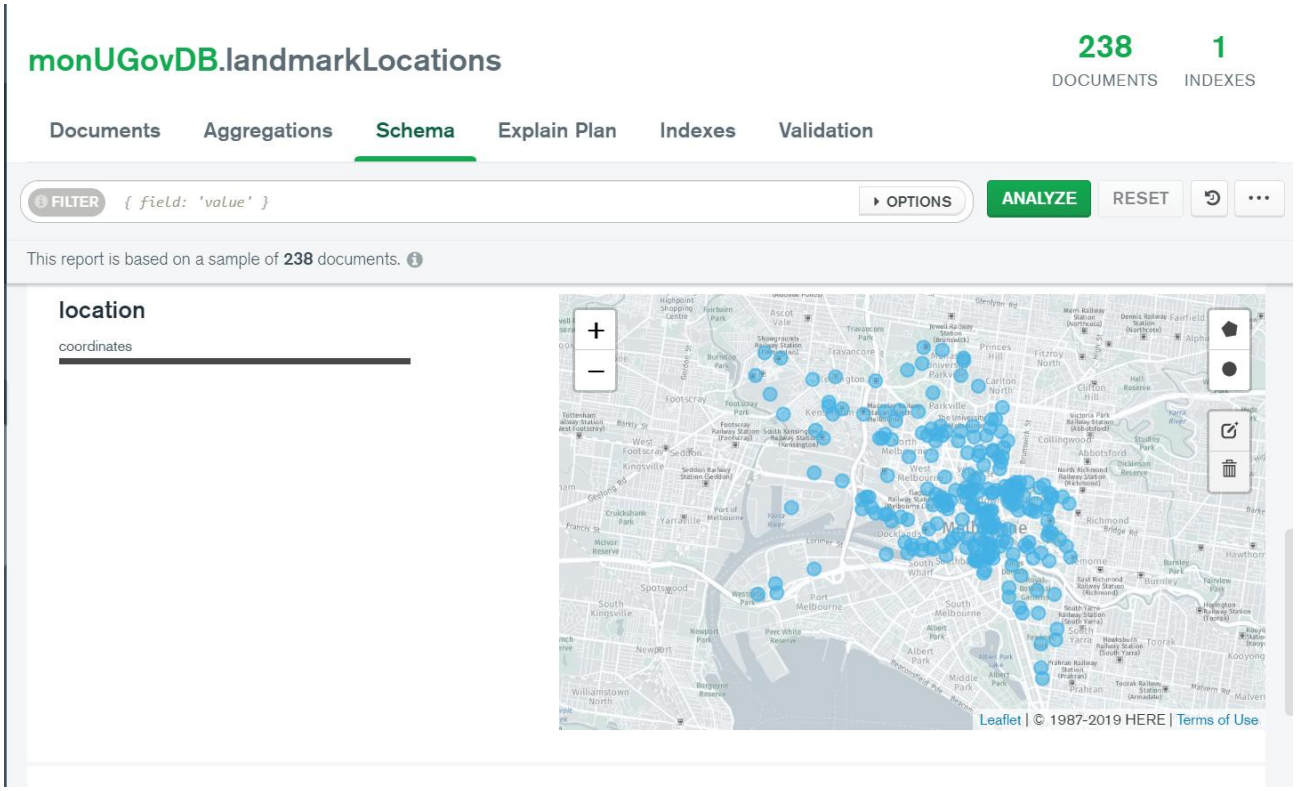
```
db.landmarks.aggregate([{$set:{location:{type:"Point",coordinates:["$lon","$lat"]}}},{$project:{"_id":0,"landmarkName":1,"category":1,"theme":1,"location":1}},{$out:"landmarkLocations"}])
```

SCREENSHOTS:



(ii) Using MongoDB Compass provide a screenshot of the updated geojson point locations on a map visualisation.

SCREENSHOT:



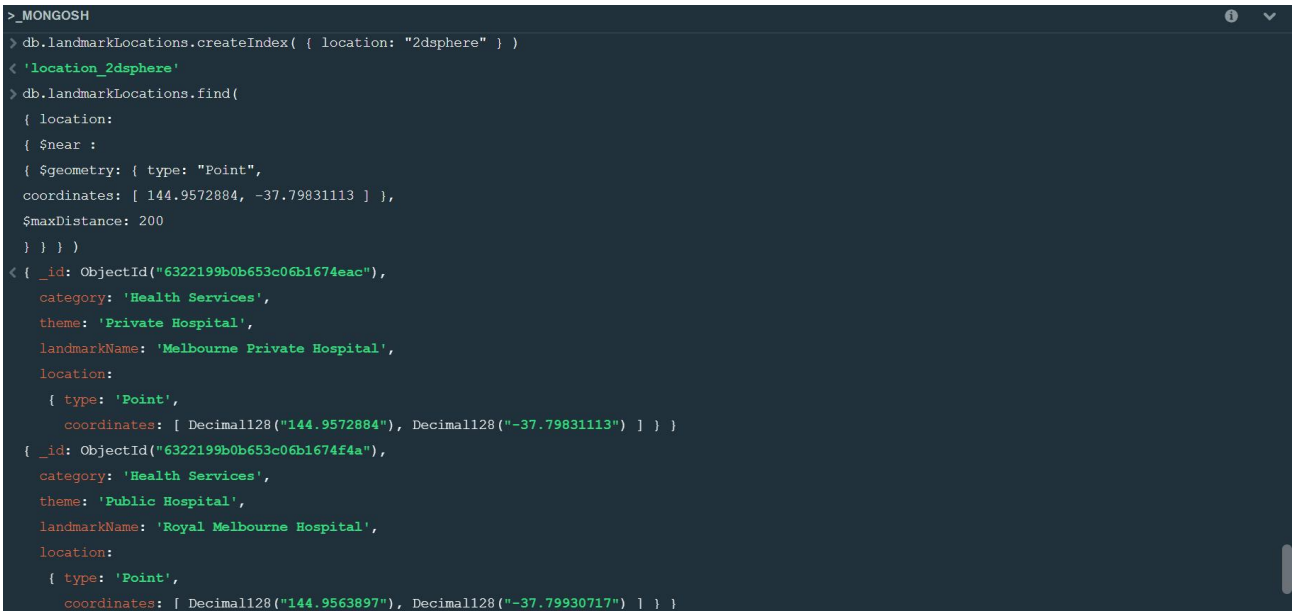
(iii) List all landmark names within 200m of "Melbourne Private Hospital". Provide the MongoShell code and a screenshot of the output containing the landmark names.

Note: *you can split up the query into 2 parts for this task.*

CODE:

```
db.landmarkLocations.createIndex( { location: "2dsphere" } )

db.landmarkLocations.find(

{ location:

{ $near :

{ $geometry: { type: "Point",

coordinates: [ 144.9572884, -37.79831113 ] },

$maxDistance: 200

} } } )
```

SCREENSHOT:



C.2. Analysis using `Cassandra`.

***Data Requirement*:**	The data for this task is contained in the following files: • `suburbs.csv` • `landmarks.csv` • `properties.csv`
Software Requirement:	(i) A software that can run Cassandra Shell (cqlsh) commands (e.g. VS Code, Terminal, Command Prompt etc.)
Overall Task C.2. Outputs:	(i) Screenshots added to the reports for each query, with before and after screenshots for any insert/update/delete made to the database. (ii) All code added in a properly commented file named **C2_Cassandra.cql**.

Task Requirements:

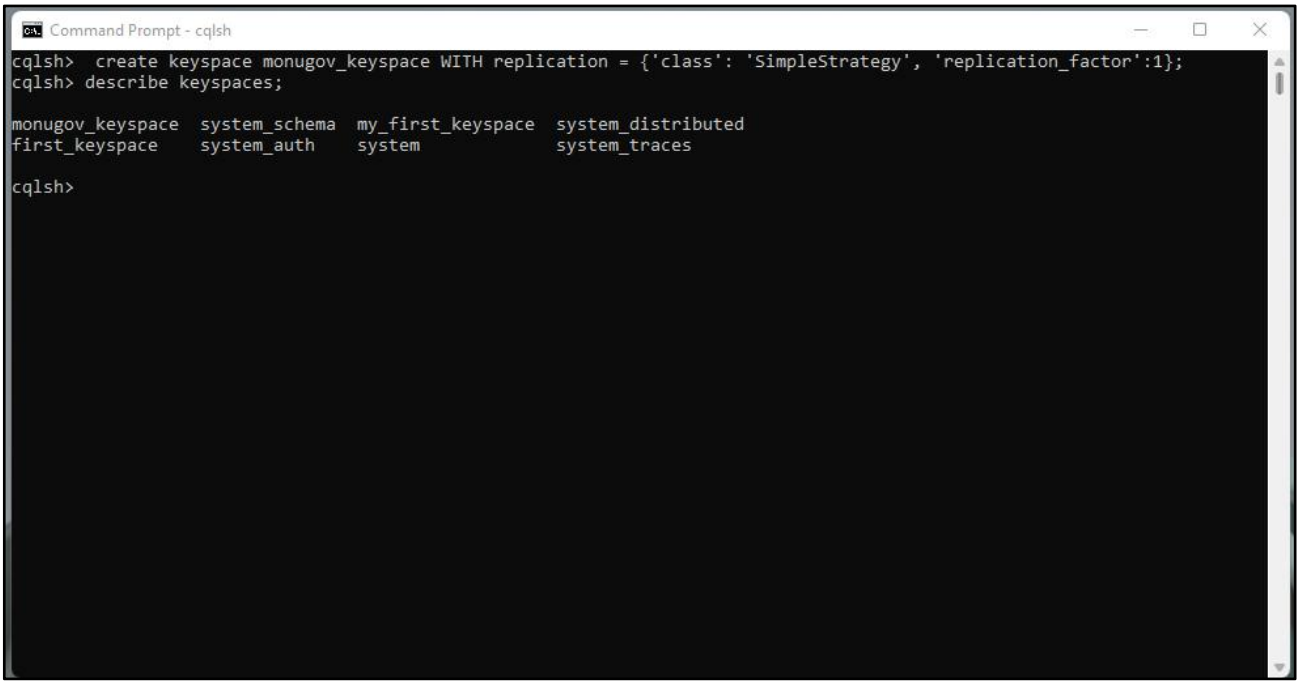
C.2.1. Using Cassandra Shell (cqlsh) creates a keyspace called **monugov_keyspace** for the Cassandra database, with SimpleStrategy and replication factor of 1.

Provide in your report the code and a screenshot of the created keyspace in the **list of all keyspace**.

CODE:

```
create keyspace monugov_keyspace WITH replication = {'class': 'SimpleStrategy',
'replication_factor':1};
```

SCREENSHOT:



C.2.2. Using the Cassandra **COPY** command import the data from into the

`monugov_keyspace` using the following tables:
`suburbs.csv` into the `suburbs` table
`landmarks.csv` into the `landmarks` table
`properties.csv` into the `properties` table

Note: *You may be required to create more tables and data types to support the queries.*
Provide in your report the code and a screenshot of the created tables in the **list of all tables**.

CODE:

Suburb Table:
CREATE TABLE suburbs (id text, council_area text, suburb text, region_name text, postcode int, property_count int, PRIMARY KEY (suburb, id));

COPY suburbs (id,council_area, suburb, region_name, postcode, property_count) FROM 'G:\My Drive\2022\Semester_2_(2022)\FIT3176 (Advanced Database Design)\Assignment 1\Assignment 1 Data\suburbs.csv' WITH HEADER = TRUE AND DELIMITER=',';

Landmarks Table:
CREATE TABLE landmarks (id text, category text, theme text, landmark_name text, latitude float, longitude float, house_number text, street text, suburb text, postcode int, PRIMARY KEY((id),landmark_name));

COPY landmarks (id,category, theme, landmark_name, latitude, longitude, house_number, street, suburb, postcode) FROM 'G:\My Drive\2022\Semester_2_(2022)\FIT3176 (Advanced Database Design)\Assignment 1\Assignment 1 Data\landmarks.csv' WITH HEADER = TRUE AND DELIMITER=',';

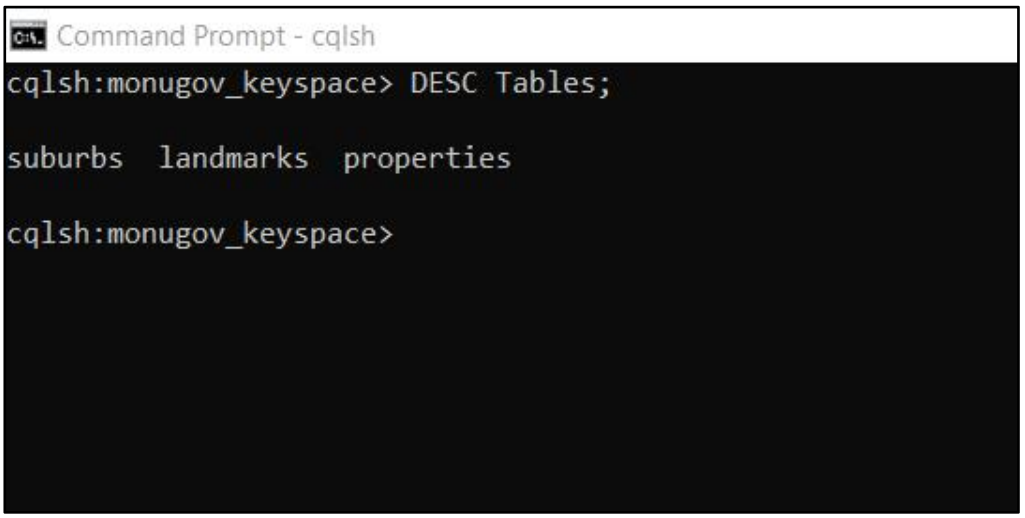
Properties Table:
//Requires UDT of history:
CREATE TYPE history (sold_by text, date date, price int);

CREATE TABLE properties (id text, address text, postcode int, property_history list<frozen<history>>, rooms int, suburb text, type_id text, PRIMARY KEY(id,type_id));

COPY properties (id,address,postcode,property_history, rooms, suburb, type_id) FROM 'G:\My Drive\2022\Semester_2_(2022)\FIT3176 (Advanced Database Design)\Assignment 1\Assignment 1 Data\properties.csv' WITH HEADER = TRUE AND DELIMITER='|';

SCREENSHOT:

All tables shown:



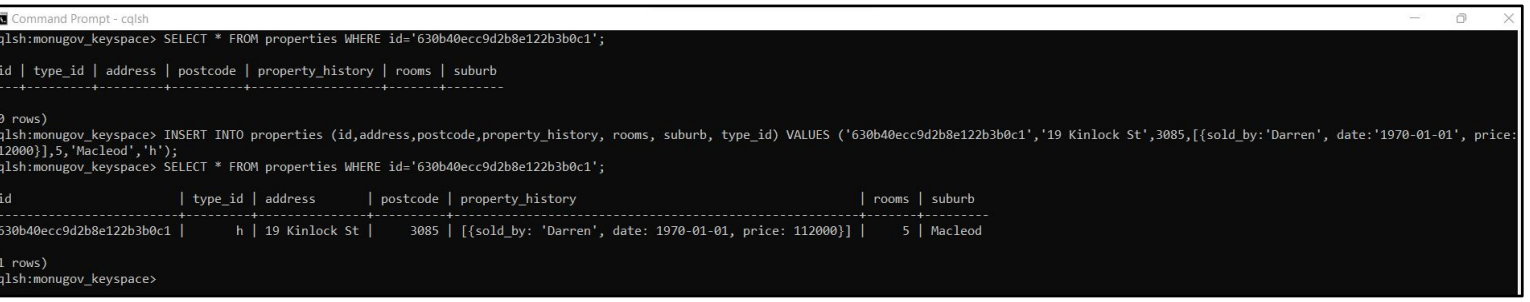

```
cqlsh:monugov_keyspace> CREATE TYPE history (sold_by text, date date, price int);
cqlsh:monugov_keyspace> desc type history;

CREATE TYPE monugov_keyspace.history (
    sold_by text,
    date date,
    price int
);
cqlsh:monugov_keyspace>
```

C.2.3. Insert the following property data related into the appropriate tables:

address	19 Kinlock St
suburb	Macleod
postcode	3085
region name	Northern Metropolitan
property count	4168
council area	Banyule City Council
rooms	5
type	house (ENTERED AS h)
price	1120000
date	1970-01-01T00:00:00Z (Red section ignored based on Ed Post)
sold by landmark in the same suburb	Darren Gresswell Theatre is of category ‘Place Of Assembly’ with a theme of ‘Theatre Live’ located at (lat: -37.712422, lon: 145.072617) address (house no and street): 1 Forrest Road

Provide in your report the insert code and a screenshot of the inserted data by using SELECTs.

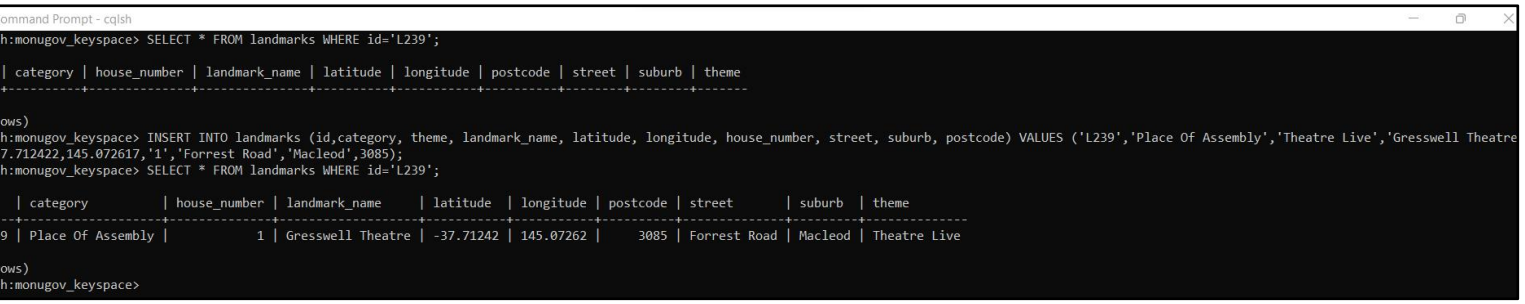
PROPERTIES TABLE:



SUBURBS TABLE:

No data inserted. The suburb Macleod is already contained in the suburbs table.

LANDMARKS TABLE:



C.2.4.

(i) Using the `suburbs` table, find the row that contains the `suburb` called `Caulfield`.

```
Command Prompt - cqlsh
cqlsh:monugov_keyspace>     SELECT * FROM suburbs WHERE suburb = 'Caulfield';

 suburb    | id   | council_area           | postcode | property_count | region_name
-----------+------+------------------------+----------+----------------+----------------------
 Caulfield | S272 | Glen Eira City Council |     3162 |           2379 | Southern Metropolitan

(1 rows)
cqlsh:monugov_keyspace>
```

(ii) Now update the table to add:
one new column called `otherHomeGround` with value `true` and another new column called `team` with value `yourGroupNo`.

- Added new columns:

```
Command Prompt - cqlsh
lsh:monugov_keyspace> Describe suburbs;

EATE TABLE monugov_keyspace.suburbs (
    suburb text,
    id text,
    council_area text,
    postcode int,
    property_count int,
    region_name text,
    PRIMARY KEY (suburb, id)
) WITH CLUSTERING ORDER BY (id ASC)
    AND bloom_filter_fp_chance = 0.01
    AND caching = {'keys': 'ALL', 'rows_per_partition': 'NONE'}
    AND comment = ''
    AND compaction = {'class': 'org.apache.cassandra.db.compaction.SizeTieredCompactionStrategy', 'max_threshold': '32', 'min_threshold': '4'}
    AND compression = {'chunk_length_in_kb': '64', 'class': 'org.apache.cassandra.io.compress.LZ4Compressor'}
    AND crc_check_chance = 1.0
    AND dclocal_read_repair_chance = 0.1
    AND default_time_to_live = 0
    AND gc_grace_seconds = 864000
    AND max_index_interval = 2048
    AND memtable_flush_period_in_ms = 0
    AND min_index_interval = 128
    AND read_repair_chance = 0.0
    AND speculative_retry = '99PERCENTILE';

lsh:monugov_keyspace> ALTER TABLE suburbs ADD (other_home_ground boolean, team int);
lsh:monugov_keyspace> Describe suburbs;

EATE TABLE monugov_keyspace.suburbs (
    suburb text,
    id text,
    council_area text,
    other_home_ground boolean,
    postcode int,
    property_count int,
    region_name text,
    team int,
    PRIMARY KEY (suburb, id)
) WITH CLUSTERING ORDER BY (id ASC)
    AND bloom_filter_fp_chance = 0.01
    AND caching = {'keys': 'ALL', 'rows_per_partition': 'NONE'}
    AND comment = ''
    AND compaction = {'class': 'org.apache.cassandra.db.compaction.SizeTieredCompactionStrategy', 'max_threshold': '32', 'min_threshold': '4'}
    AND compression = {'chunk_length_in_kb': '64', 'class': 'org.apache.cassandra.io.compress.LZ4Compressor'}
    AND crc_check_chance = 1.0
    AND dclocal_read_repair_chance = 0.1
    AND default_time_to_live = 0
    AND gc_grace_seconds = 864000
```

```
lsh:monugov_keyspace>     SELECT * FROM suburbs;

uburb             | id    | council_area                  | other_home_ground | postcode | property_count | region_name                | team
------------------+-------+-------------------------------+-------------------+----------+----------------+----------------------------+------
       Deer Park  |  S74  |          Brimbank City Council |              null |     3023 |           6388 |       Western Metropolitan | null
  West Melbourne  | S219  |         Melbourne City Council |              null |     3003 |           2230 |      Northern Metropolitan | null
      Heidelberg  | S114  |           Banyule City Council |              null |     3084 |           2890 |       Eastern Metropolitan | null
         Croydon  |  S68  |         Maroondah City Council |              null |     3136 |          11925 |       Eastern Metropolitan | null
    Woori Yallock | S333  |     Yarra Ranges Shire Council |              null |     3139 |           1164 |          Northern Victoria | null
   Brighton East  |  S35  |           Bayside City Council |              null |     3187 |           6938 |      Southern Metropolitan | null
Cranbourne South  | S371  |             Casey City Council |              null |     3977 |            615 | South-Eastern Metropolitan | null
        Essendon  |  S90  |     Moonee Valley City Council |              null |     3040 |           9264 |       Western Metropolitan | null
        Kurunjang | S133  |           Melton City Council |              null |     3337 |           3553 |          Northern Victoria | null
      Forest Hill |  S99  |        Whitehorse City Council |              null |     3131 |           4385 |       Eastern Metropolitan | null
      Maribyrnong | S139  |       Maribyrnong City Council |              null |     3032 |           4918 |       Western Metropolitan | null
        Eynesbury | S318  |           Melton City Council |              null |     3338 |            852 |           Western Victoria | null
         Richmond | S178  |           Yarra City Council  |              null |     3121 |          14949 |      Northern Metropolitan | null
        Edithvale |  S83  |         Kingston City Council |              null |     3196 |           2546 | South-Eastern Metropolitan | null
  Blackburn South |  S29  |        Whitehorse City Council |              null |     3130 |           4387 |       Eastern Metropolitan | null
```

- Set the values of the columns

```
Command Prompt - cqlsh
lsh:monugov_keyspace> SELECT * FROM suburbs WHERE suburb = 'Caulfield';

uburb     | id   | council_area           | other_home_ground | postcode | property_count | region_name           | team
----------+------+------------------------+-------------------+----------+----------------+-----------------------+------
aulfield | S272 | Glen Eira City Council |              null |     3162 |           2379 | Southern Metropolitan | null

 rows)
lsh:monugov_keyspace> UPDATE suburbs SET other_home_ground = true, team = 29 WHERE suburb ='Caulfield' AND id ='S272';
lsh:monugov_keyspace> SELECT * FROM suburbs WHERE suburb = 'Caulfield';

uburb     | id   | council_area           | other_home_ground | postcode | property_count | region_name           | team
----------+------+------------------------+-------------------+----------+----------------+-----------------------+------
aulfield | S272 | Glen Eira City Council |              True |     3162 |           2379 | Southern Metropolitan |   29

 rows)
lsh:monugov_keyspace>
```

(iii) Make sure the data in both columns is stored in cassandra for 300000 seconds. ***Note:*** *the* `yourGroupNo` *will be different for each group attempting the FIT3176 assignment.*

- Set the TTL for both columns to 300000 seconds:
- Shows the remaining living time.

```
Command Prompt - cqlsh
glsh:monugov_keyspace> UPDATE suburbs USING TTL 300000 SET other_home_ground = true, team = 29 WHERE suburb ='Caulfield' AND id ='S272';
glsh:monugov_keyspace> SELECT id, suburb, TTL(other_home_ground), TTL (team) FROM suburbs WHERE suburb ='Caulfield' AND id ='S272';

d    | suburb    | ttl(other_home_ground) | ttl(team)
-----+-----------+------------------------+-----------
S272 | Caulfield |                 299920 |    299920

 rows)
glsh:monugov_keyspace> SELECT id, suburb, writetime(other_home_ground), writetime(team) FROM suburbs;
```

- Values have been set:

```
Command Prompt - cqlsh
glsh:monugov_keyspace> SELECT * FROM suburbs WHERE suburb = 'Caulfield';

suburb    | id   | council_area           | other_home_ground | postcode | property_count | region_name           | team
----------+------+------------------------+-------------------+----------+----------------+-----------------------+------
Caulfield | S272 | Glen Eira City Council |              True |     3162 |           2379 | Southern Metropolitan |   29

 rows)
glsh:monugov_keyspace>
```

- Below shows the timestamp of the updated columns for the ‘Caulfield’ suburb:

id	suburb	writetime(other_home_ground)	writetime(team)
S194	Spotswood	null	null
S111	Hampton	null	null
S43	Burwood East	null	null
S272	Caulfield	1663062586495000	1663062586495000
S7	Alphington	null	null
S107	Gowanbrae	null	null
S128	Kew	null	null
S66	Cranbourne	null	null
S129	Kew East	null	null
S287	North Warrandyte	null	null
S186	Sandringham	null	null
S352	Brookfield	null	null
S102	Frankston South	null	null
S312	Derrimut	null	null
S5	Albert Park	null	null
S138	Malvern East	null	null
S213	Vermont South	null	null
S126	Keilor East	null	null
S244	Healesville	null	null
S254	Montrose	null	null
S116	Heidelberg West	null	null
S197	St Kilda	null	null
S176	Preston	null	null
S19	Balwyn	null	null

Provide in your report the insert code and a screenshot of the updated data by using SELECTs before and after updating, showing the **remaining living time** and **timestamps** for the updated columns.

C.2.5. Read the queries from C2.7 (There is no C2.7 (should be 2.6)) and create 1 secondary index to help speed up the queries. Provide in your report the code to create the index and screenshots of the created index from cql.

To speed up the first query, I have created a secondary index on type_id field.

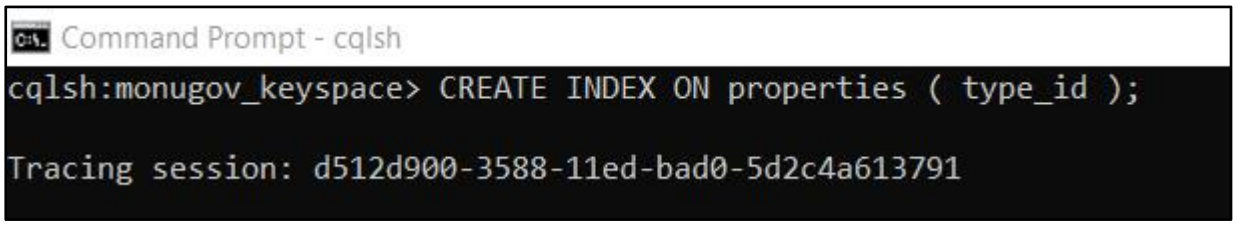
CODE:

CREATE INDEX ON properties (type_id);

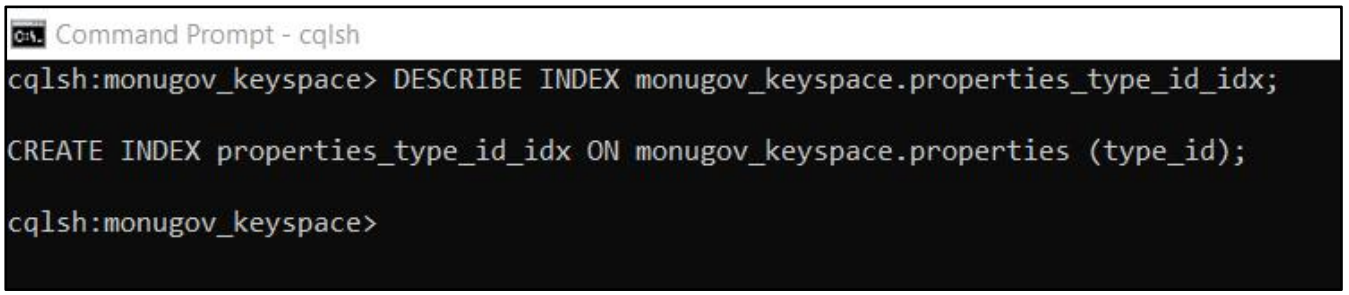
DESCRIBE INDEX monugov_keyspace.properties_type_id_idx;

SCREENSHOTS:

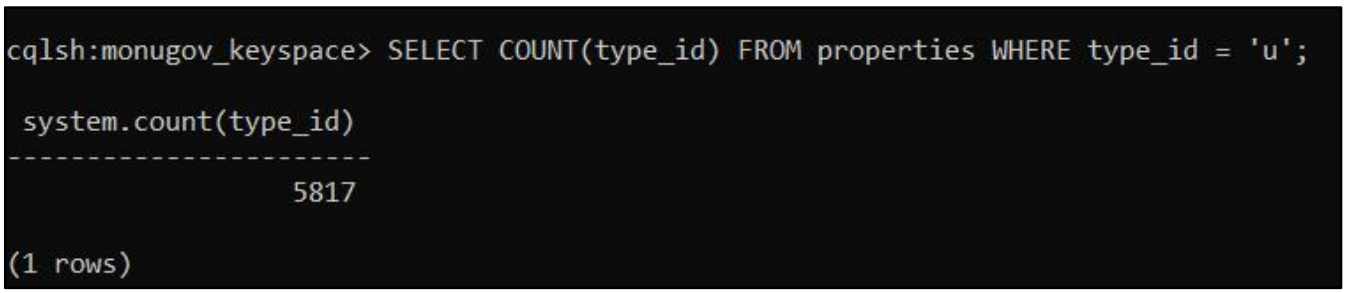
- Image below shows the command to create the index.



- The output shows the index has been created.



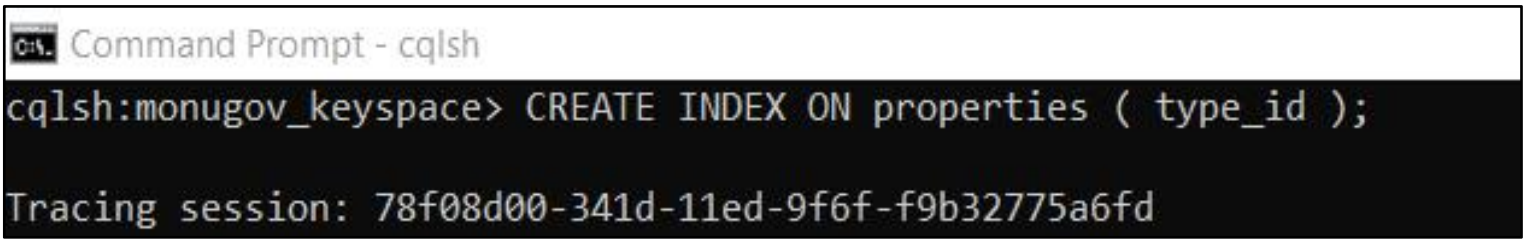
- Image below shows the use of the index in the the query.

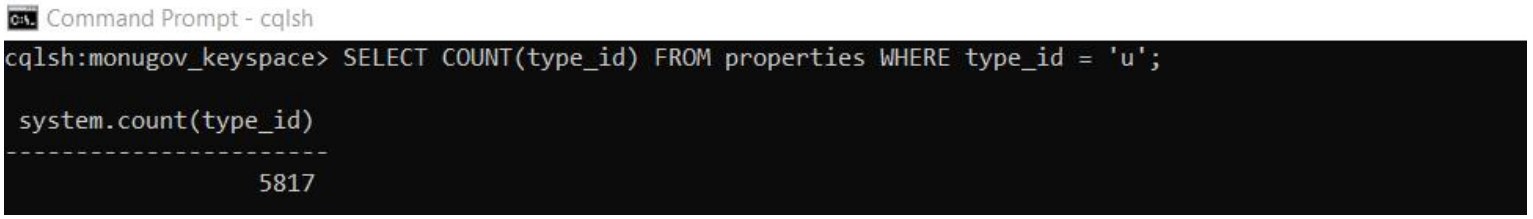


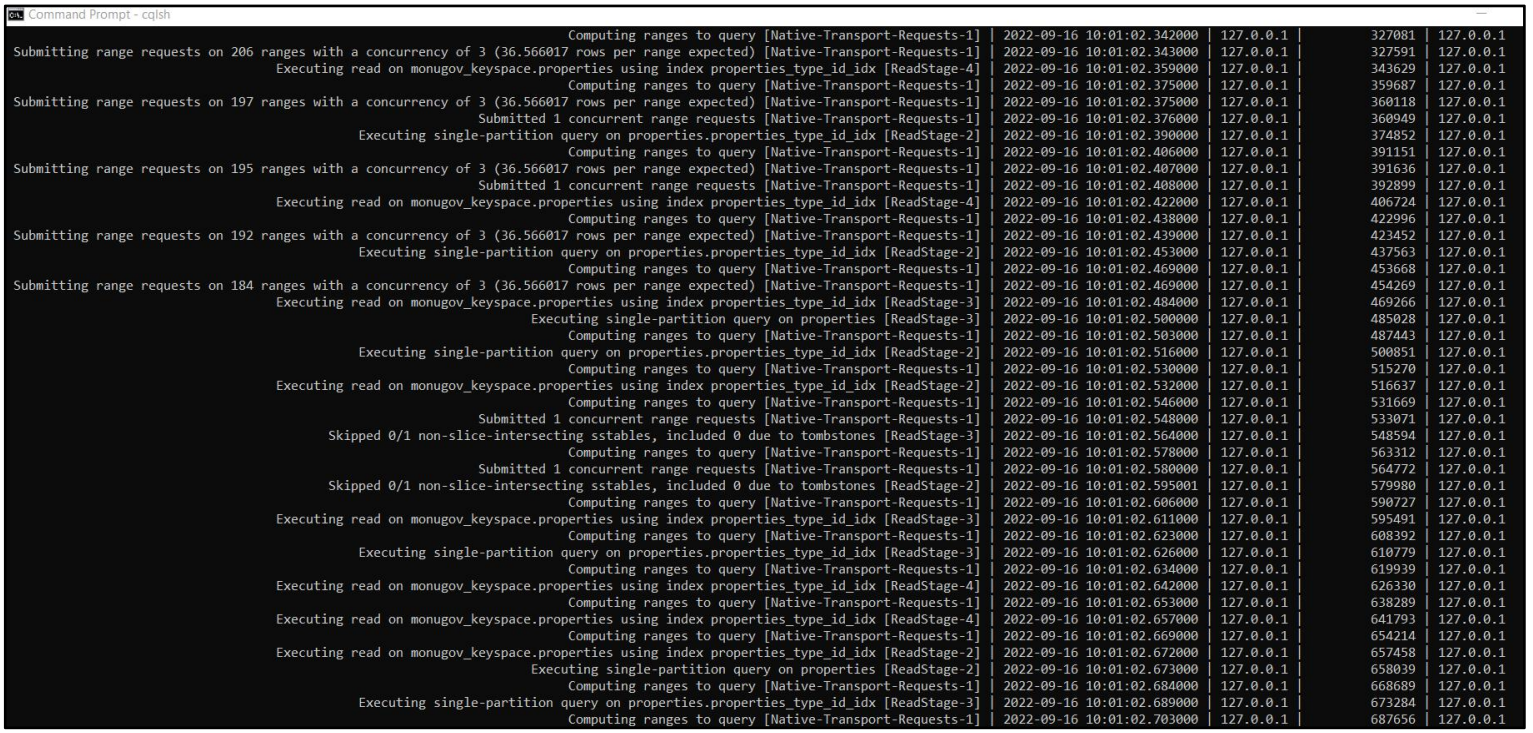
C.2.6. Use Cassandra shell to answer the following queries and in addition to the screenshots required for each question, provide the time taken while executing each query:

Note: *You can create any number of commands, tables/column families, indices etc. to answer your queries for this section. However, marks for this section depend on the query efficiency e.g. the storage, processing time etc. Therefore, using more than the required amount of tables, queries, indices to answer a section may incur mark penalties.*

(i) Find how many properties of type units (i.e u) were in the database. Provide the cql code to find the count and a screenshot containing the property type and the count.







(ii) List all of the landmarks with a postcode greater than (but not equal to) 3200. Provide the cql code and a screenshot containing two landmark names and the postcode ordered by the ascending

alphabetical order of landmark name.

CODE:

CREATE TABLE landmarks_by_postcode (id text, category text, theme text, landmark_name text, latitude float, longitude float, house_number text, street text, suburb text, postcode int, PRIMARY KEY((postcode),landmark_name));

COPY landmarks_by_postcode (id,category, theme, landmark_name, latitude, longitude, house_number, street, suburb, postcode) FROM 'G:\My Drive\2022\Semester_2_(2022)\FIT3176 (Advanced Database Design)\Assignment 1\Assignment 1 Data\landmarks.csv' WITH HEADER = TRUE AND DELIMITER=',';

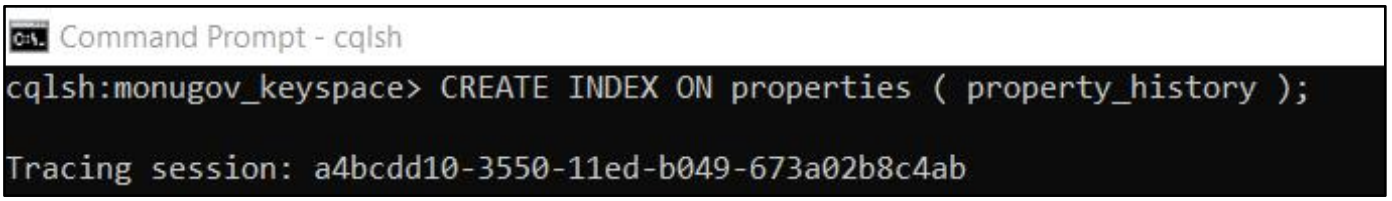
SELECT postcode, landmark_name FROM landmarks_by_postcode WHERE postcode > 3200 allow filtering;

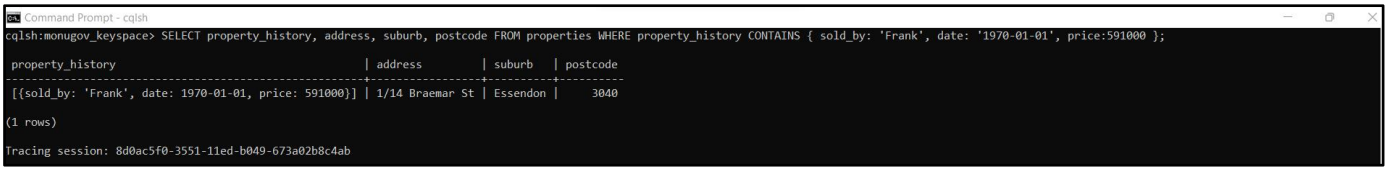
SCREENSHOT:

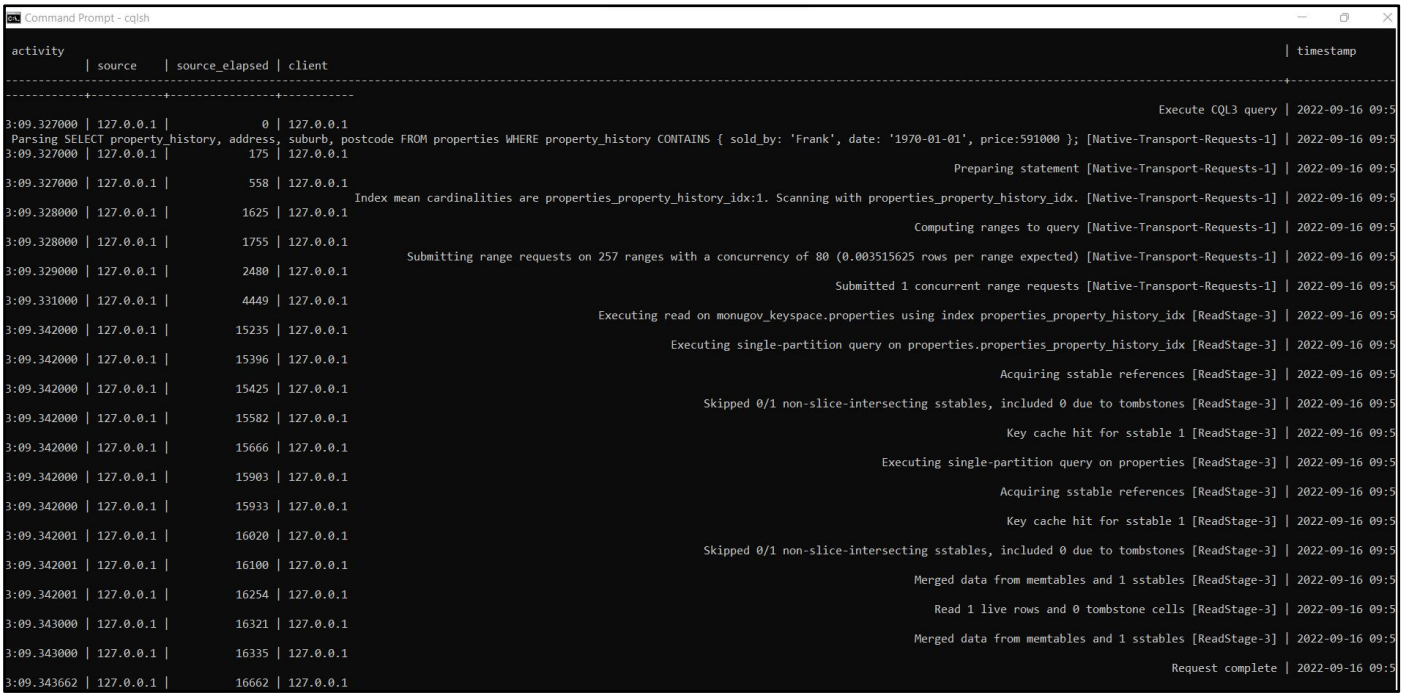
```
cqlsh:monugov_keyspace> SELECT postcode, landmark_name FROM landmarks_by_postcode WHERE postcode > 3200 allow filtering;

 postcode | landmark_name
----------+-----------------------------------------
     3207 |                                    Kraft
     3207 | Melbourne International Karting Complex
```

```
Tracing session: 7b42d230-3561-11ed-b049-673a02b8c4ab

 activity                                                                                                                    | timestamp                  | source    | source_elapsed | client
-----------------------------------------------------------------------------------------------------------------------------+----------------------------+-----------+----------------+-----------
                                                                                                          Execute CQL3 query | 2022-09-16 11:47:11.443000 | 127.0.0.1 |              0 | 127.0.0.1
 Parsing SELECT postcode, landmark_name FROM landmarks_by_postcode WHERE postcode > 3200 allow filtering; [Native-Transport-Requests-1] | 2022-09-16 11:47:11.443000 | 127.0.0.1 |            123 | 127.0.0.1
                                                                           Preparing statement [Native-Transport-Requests-1] | 2022-09-16 11:47:11.443000 | 127.0.0.1 |            299 | 127.0.0.1
                                                                      Computing ranges to query [Native-Transport-Requests-1] | 2022-09-16 11:47:11.443000 | 127.0.0.1 |            814 | 127.0.0.1
         Submitting range requests on 257 ranges with a concurrency of 1 (0.0 rows per range expected) [Native-Transport-Requests-1] | 2022-09-16 11:47:11.443000 | 127.0.0.1 |            958 | 127.0.0.1
                                                              Submitted 1 concurrent range requests [Native-Transport-Requests-1] | 2022-09-16 11:47:11.444000 | 127.0.0.1 |           1579 | 127.0.0.1
                       Executing seq scan across 0 sstables for (min(-9223372036854775808), min(-9223372036854775808)] [ReadStage-3] | 2022-09-16 11:47:11.446000 | 127.0.0.1 |           2920 | 127.0.0.1
                                                                    Read 17 live rows and 0 tombstone cells [ReadStage-3] | 2022-09-16 11:47:11.446000 | 127.0.0.1 |           3228 | 127.0.0.1
                                                                                                            Request complete | 2022-09-16 11:47:11.446465 | 127.0.0.1 |           3465 | 127.0.0.1


cqlsh:monugov_keyspace>
```

(iii) Which properties were sold by 'Frank' on '1970-01-01' with a price of 591000? Provide the cql code and a screenshot containing the property history, address, suburb, postcode.







C.3. Polyglot Persistence (Non-Coding Section)

In addition to the previously provided data:

```
suburbs.csv
landmarks.csv
properties.json
```

MonUGov has also collected additional data from the data.melbourne.vic.gov.au website such as: Urban Forest, Environmental resources consumed and Cafes and restaurants. ***Note**: You are not required to download the additional data.*

For future analysis of finding housing options according to their client's preferences in budget, property type, area, location etc, MonUGov wants to create a data store architecture to collect and store the data.

After seeing the capabilities of both MongoDB and Cassandra. MonUGov has requested you create a Polyglot Persistence approach using a Relational Database (Oracle), a Document-Oriented Database (MongoDB) and a Column-Oriented Database (Cassandra) to store the data.

Task Requirement:

***Note**: In this section*
data may refer to: suburbs.csv, landmarks.csv, properties.json, Urban Forest data, Environmental resources consumed and Cafes and restaurants.
database may refer to Relational Database (Oracle), Document-Oriented Database (MongoDB) and Column-Oriented Database (Cassandra)

C.3.1. Specify which data will be stored in which type of database:

(i) in your own words provide a brief paragraph to explain how each type of database would be connected to MonUGov's data store architecture and why the particular database would be helpful to store the particular data.

Ans:

Since Polyglot Data Architecture is decentralized, it actually allows integration of various/different, separate networks of databases and storage of large amounts of different data to achieve optimal performance while meeting most, if not all business requirements. As a result, depending on the operations the Architecture is dealing with, it can simply refer to the relevant database to perform it, thus promoting flexibility. For instance, when dealing with operations involving interrelated/interdependent collection of data, the Oracle database can be referred without sacrificing speed, performance or other databases.

(ii) with the help of your selected examples provide a comparison in a tabular format with details on the main strengths and weaknesses of each database when it comes to storing the data provided by MonUGov.

Ans:

	Relational Database	**Document-Oriented Database**	**Column-Oriented Database**
Strengths	- Easy querying process, thus it wouldn't be difficult for MonUGov to perform CRUD operations. - Promotes ACID, which is critical to achieve integrity whilst dealing with large amounts of data/information, something MonUGov will definitely have to deal with.	- It is designed to deal with various unrelated/independent data/information, which is the case here (e.g. dealing with operations involving both landmarks.csv and urban forest data). - Schema-less, which is great for retaining large amounts of data/information of various structural formats/states (for instance the urban forest data has more columns compared to other documents, with fields such as "Useful Life Expectancy" and "Genus").	- Storage-efficient due to being excellent at compressing data/information. This is extremely important for files with large amounts of fields, such as the Urban Forest Data. - Fast in terms of querying and load times, which will be beneficial for MonUGov who wish to deal with large amounts of data.
Weaknesses	- Since MonUGov will be dealing with large amounts of data/information, there might be a problem in terms of database maintenance. - Dealing with large amount of data translates to requirements of large physical memory.	- Does not observe ACID, meaning that there might be inconsistencies in data/information. - Being non-ACID also translates to non-atomicity, meaning that faulty/error-prone operations might affect the entire database on large magnitudes.	- Column-oriented databases are inefficient in terms of updating transactions, as its main capability/feature is to perform analysis/data retrieval. - While it can deal with row-oriented/specific queries, it requires some time to process/complete the operation.

References:

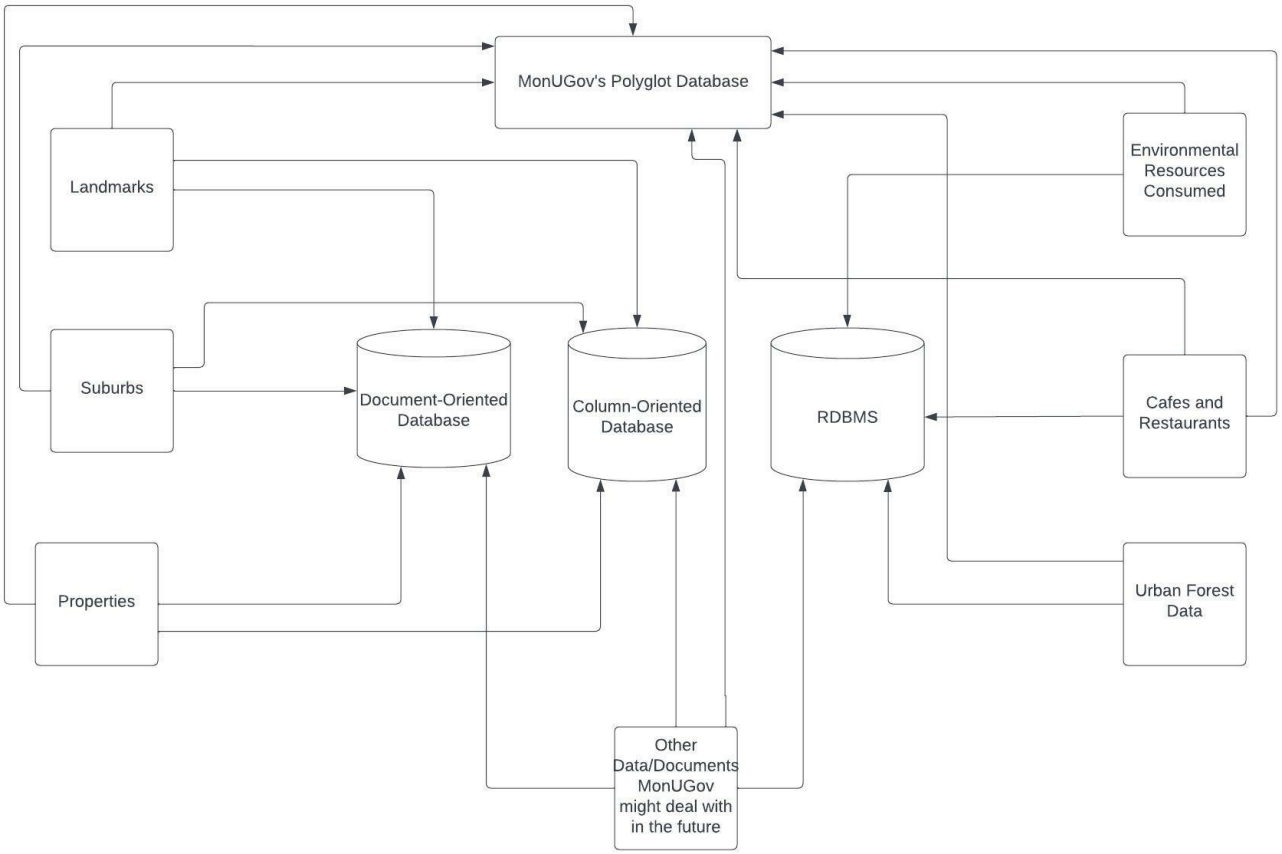
DatabaseTown. 2022. *Relational Database Benefits and Limitations (Advantages & Disadvantages) -DatabaseTown*. [online] Available at: <https://databasetown.com/relational-database-benefits-and-limitations/#:~:text=The%20main%20benefits%20of%20using,issue%20of%20speed%20can%20arise.>.

Williams, A., 2022. *Document Database {Definition, Features, Use Cases}*. [online] Knowledge Base by phoenixNAP. Available at: <https://phoenixnap.com/kb/document-database>.

Williams, A., 2021. *NoSQL database types explained: Column-oriented databases*. [online] SearchDataManagement. Available at: <https://www.techtarget.com/searchdatamanagement/tip/NoSQL-database-types-explained-Column-oriented-databases>.

Task Requirement:

C.3.2. Draw a data architecture diagram to illustrate how Polyglot Persistence would be implemented in C3.1.



Overall Task C.3. Outputs:

- ❒ A report specifying the requirements in Task C.3.1. and Task C.3.2 with any references made from any sources using a proper referring style.

Note: *Penalties may be applied for any generic explanations not specific to your examples (for C.3.1.) and to* **`monUGovDB`***.(for C.3.2.). This section does not require any code submissions.* ***Mark penalties are applicable if correct referencing is not provided.***

C.4. Connecting to Drivers (Optional Bonus Section - up to 5 marks)

Note: *This section is for those who are looking for a real challenge. Even without completing this section, there is a possibility of scoring full marks for the assignment. However, to attain full marks for Task C.4., both of your scripts must be runnable in the MacOS/Windows terminal/command prompt. Therefore, attempt at your own risk :)*

***Data Requirement*:**	Same as Task C.1 and Task C.2.
Software Requirement:	(i) A software that can run scripts such as Python script commands (e.g. Pycharm etc.).
Overall Task C.4. Outputs:	(i) List of steps to take in order to connect MongoDB and Cassandra drivers. (ii) Two properly commented driver script files i.e. one for MongoDB (named e.g. C4_MongoDB.py) and another for Cassandra (named e.g. C4_Cassandra.py).

After reading your reports in Task C.1. and C1.2, monUGovDB now has more knowledge about the NoSQL databases: MongoDB and Cassandra. They also came to know that these databases can be incorporated into applications and be used as backend database stores.

So, monUGovDB has asked your team to create a **runnable script incorporating the code from Task C.1. and C.2. into a Python application with the help of drivers.**

Note: *For details about how to use the MongoDB and Cassandra drivers you can refer to the steps given in the Lectures.*

Task Requirement:

(i) Provide in your report a list of steps you have taken to connect the MongoDB and Cassandra drivers

NOT ATTEMPTED

(ii) Convert the code from Task C.1. and C.2. into runnable applications using script files (e.g. .py files) and **not Jupyter notebook files.**

NOT ATTEMPTED