1. create a database called: FIT5137A1MRDB
    a. use FIT5137A1MRDB
        i.
```
> use FIT5137A1MRDB
switched to db FIT5137A1MRDB
>
```
2. Mongo Shell command(s), create the following 2 collections userProfiles and placeProfiles
    a. db.createCollection("userProfiles")
        i.
```
> db.createCollection("userProfiles")
{ "ok" : 1 }
```
    b. db.createCollection("placeProfiles")
        i.
```
> db.createCollection("placeProfiles")
{ "ok" : 1 }
```
3. Importing data using MongoDB Compass and appropriate MongoDB Data Types
    a. userProfile.json into the userProfiles collection
        i.

```
_id: 1001
favCuisines: "American"
favPaymentMethod: "cash"
location: Object
    latitude: 22.139997
    longitude: -100.9788
otherDemographics: Object
    employment: "student"
    religion: "none"
personalTraits: Object
    birthYear: 1989
    height: 1.77
    maritalStatus: "single"
    weight: 69
personality: Object
    drinkLevel: "abstemious"
    favColor: "black"
    interest: "variety"
    typeOfWorker: "thrifty-protector"
preferences: Object
    ambience: "family"
    budget: "medium"
    dressPreference: "informal"
    smoker: false
    transport: "on foot"
```

    b. placeProfiles.json into the placeProfiles collection
        i.

```
_id: 132560
acceptedPaymentModes: "cash"
address: Object
    city: "victoria"
    country: "Mexico"
    state: "tamaulipas"
    street: "frente al tecnologico"
cuisines: "Regional"
location: Object
    latitude: 23.7523041
    longitude: -99.166913
parkingArragements: "public"
placeFeatures: Object
    accessibility: "no_accessibility"
    alcohol: "No_Alcohol_Served"
    area: "open"
    dressCode: "informal"
    franchise: "f"
    otherServices: "none"
    price: "low"
    smokingArea: "permitted"
placeName: "puesto de gorditas"
```

4. Two data modelling methods, which are embedding and referencing

a. Embedding is the process of attaching all the data in one document instead of multiple documents known as normalized data model where referencing means that storing the data in multiple documents with ensuring a reference to link them together, similar to the foreign key in relational databases known as normalized data model.
b. Importing openingHours.csv into your FIT5137A1MRDB database. At the beginning I had to create a table openingHours

db.createCollection("openingHours")

i.

```
> db.createCollection("openingHours")
{ "ok" : 1 }
```

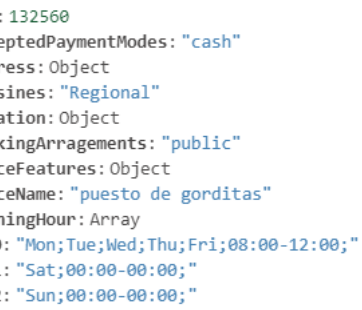
```
_id: ObjectId("5f7b05c1c6c41a2f3081954f")
placeID: 135111
hours: "00:00-23:30;"
days: "Mon;Tue;Wed;Thu;Fri;"
```

ii.

iii. Since we have decided to use an embedding data model and we need to embed the corresponding opening hours in the placeProfiles collection so we have decided to use aggregation. All the steps have been mention here:

```
db.openingHours.aggregate([
 {

  $group: {

   _id:{placeID:"$placeID"},
   openingHour: {$push: {$concat:["$days", "$hours"]}}

   }

 },
 {
   $addFields:{
   "_id":"$_id.placeID"
   }
 },
 {
  $merge:{
   into: 'placeProfiles',
   on: '_id',
   whenMatched: 'merge',
   whenNotMatched: 'discard'
  }
 }
```

]).pretty()
After running this query we can see that openingHour array added among all the records based on similar placeId

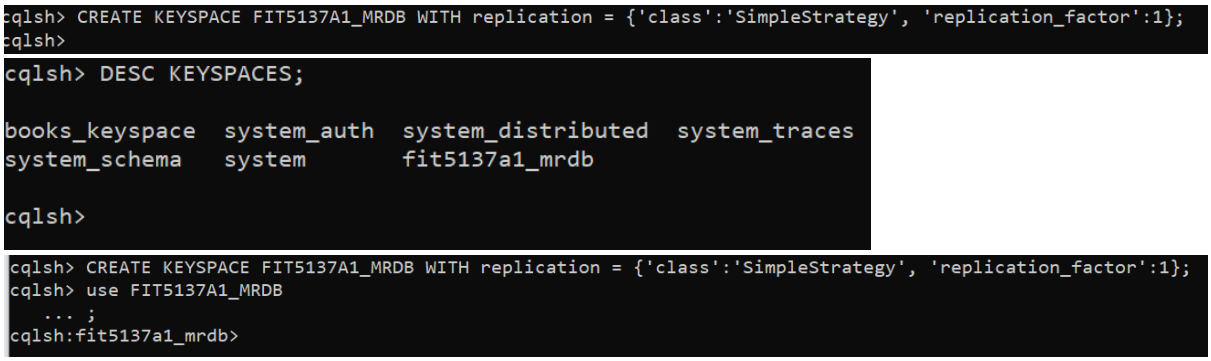


Even if we run this aggregation query several times, it would just reflect the update of openingHours into placeProfiles table.

c. The selection of the data model is an important step, and it should be aligned with objectives of the applications. Embedding data model was chosen for efficient performance, and to exploit the full potential of MongoDB functionalities. Moreover, our dataset mainly consisted of one to one relationship and one to many relationships. In this case, embedding data model is recommended. For example, one user can have either one rating or many ratings. We are required to analyze the dataset, meaning that substantial numbers of read operations will be executed. The embedding data model will facilitate the read and write operation. As a result, effective performance will be reached, and the capability to perform many operations such retrieval and update in one operation instead of multiple. Also, we are not dealing with a large dataset in our case which requires a rigorous hierarchical

5. Create a keyspace called FIT5137A1_MRDB

CREATE KEYSPACE FIT5137A1_MRDB WITH replication = {'class':'SimpleStrategy', 'replication_factor':1};



6. Create the following column families
   a. **user_ratings**

```
CREATE TYPE PersonalTraits(
    birth_year INT,
    weight INT,
    height DOUBLE,
    marital_status TEXT
);
 CREATE TYPE UserPersonality(
    interest TEXT,
    type_of_worker TEXT,
    fav_color TEXT,
    drink_level TEXT
);
CREATE TYPE UserPreferences(
    budget TEXT,
    smoker BOOLEAN,
    dress_preference TEXT,
    ambience TEXT,
    transport TEXT,
);
CREATE TYPE OtherDemographics(
    religion TEXT,
    employment TEXT,
);
CREATE TABLE user_ratings (
rating_id INT,
user_id INT,
place_id INT,
rating_place INT,
rating_food INT,
rating_service INT,
user_personal_traits FROZEN<PersonalTraits>,
user_personality FROZEN<UserPersonality>,
user_preferences FROZEN<UserPreferences>,
user_other_demographics FROZEN<OtherDemographics>,
user_fav_cuisines set<text>,
user_fav_payment_method set<text>,
PRIMARY KEY(user_id, rating_id)
);
```

```
cqlsh:fit5137a1_mrdb> DESC TABLE user_ratings;

CREATE TABLE fit5137a1_mrdb.user_ratings (
    user_id int,
    rating_id int,
    place_id int,
    rating_food int,
    rating_place int,
    rating_service int,
    user_fav_cuisines set<text>,
    user_fav_payment_method set<text>,
    user_other_demographics frozen<otherdemographics>,
    user_personal_traits frozen<personaltraits>,
    user_personality frozen<userpersonality>,
    user_preferences frozen<userpreferences>,
    PRIMARY KEY (user_id, rating_id)
) WITH CLUSTERING ORDER BY (rating_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';
```

b. place_ratings

```
CREATE TYPE PlaceAddress(
    street TEXT,
    city TEXT,
    state TEXT,
    country TEXT
);

CREATE TYPE PlaceFeatures(
    alcohol TEXT,
    smoking_area TEXT,
    dress_code TEXT,
    accessibility TEXT,
    price TEXT,
    franchise TEXT,
    area TEXT,
    other_services TEXT
);


CREATE TABLE place_ratings (
rating_id INT,
user_id INT,
place_id INT,
rating_place INT,
rating_food INT,
rating_service INT,
place_name TEXT,
place_address FROZEN<PlaceAddress>,
place_features FROZEN<PlaceFeatures>,
parking_arrangements TEXT,
accepted_payment_modes set<text>,
```

cuisines set<text>,
PRIMARY KEY(place_id, user_id, rating_id)
);

```
cqlsh:fit5137a1_mrdb> DESC TABLE place_ratings;

CREATE TABLE fit5137a1_mrdb.place_ratings (
    place_id int,
    user_id int,
    rating_id int,
    accepted_payment_modes set<text>,
    cuisines set<text>,
    parking_arrangements text,
    place_address frozen<placeaddress>,
    place_features frozen<placefeatures>,
    place_name text,
    rating_food int,
    rating_place int,
    rating_service int,
    PRIMARY KEY (place_id, user_id, rating_id)
) WITH CLUSTERING ORDER BY (user_id ASC, rating_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';
```

7. Using the cassandra COPY command import the following data into the tables:

Before running the following commands we had to include the following files place_ratings.csv and user_ratings.csv into the bin folder [apache-cassandra-3.11.7\bin]

a. place_ratings

copy place_ratings(rating_id, user_id, place_id, rating_place, rating_food, rating_service, place_name, place_address, place_features, parking_arrangements, accepted_payment_modes, cuisines) from 'place_ratings.csv' with HEADER = TRUE;

```
cqlsh:fit5137a1_mrdb> copy place_ratings(rating_id, user_id, place_id, rating_place, rating_food, rating_service, place_name, place_address, place_features, p
arking_arrangements, accepted_payment_modes, cuisines) from 'place_ratings.csv' with HEADER = TRUE;
Using 7 child processes
Processed: 1161 rows; Rate:     166 rows/s; Avg. rate:     311 rows/s
1161 rows imported from 1 files in 3.730 seconds (0 skipped).
cqlsh:fit5137a1_mrdb>
```

b. user_ratings

copy user_ratings (rating_id,user_id,place_id,rating_place,rating_food,rating_service,user_personal_traits,user_personality, user_preferences, user_other_demographics, user_fav_cuisines,user_fav_payment_method) from 'user_ratings.csv' with HEADER = TRUE;

```
cqlsh:fit5137a1_mrdb> copy user_ratings (rating_id,user_id,place_id,rating_place,rating_food,rating_service,user_personal_traits,user_personality, user_prefer
ences, user_other_demographics, user_fav_cuisines,user_fav_payment_method) from 'user_ratings.csv' with HEADER = TRUE;
Using 7 child processes
Processed: 1161 rows; Rate:     145 rows/s; Avg. rate:     275 rows/s
1161 rows imported from 1 files in 4.229 seconds (0 skipped).
cqlsh:fit5137a1_mrdb>
```

C.2. Modifying the Databases

1. Data insertion

Before insertion:

```
> db.placeProfiles.find({_id:70000}).pretty()
>
```

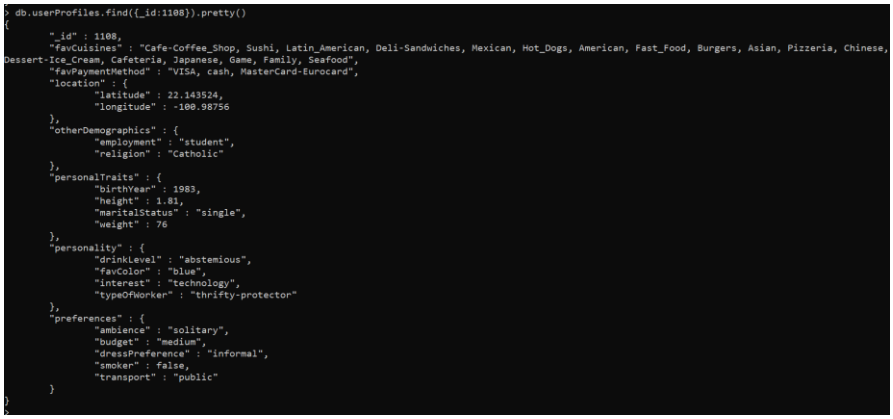
```
db.placeProfiles.insertOne({
  _id:70000,
  acceptedPaymentModes: "any",
  address: {
        city: "San Luis Potosi",
        country: "Mexico",
        state: "SLP",
        street: "Carretera Central Sn"
      },
  cuisines: "Mexican, Burgers",
  parkingArragements: "none",
  placeFeatures: {
      accessibility: "no_accessibility",
      alcohol: "No_Alcohol_Served",
      area: "closed",
      dressCode: "informal",
      franchise: "TRUE",
      otherServices: "Internet",
      price: "medium",
      smokingArea: "not permitted"
    },
    placeName: "Taco Jacks",
    openingHour: ["Mon;Tue;Wed;Thu;Fri;09:00-20:00;","Sat;12:00-18:00;","Sun;12:00-
18:00;"]})
```

After the insertion:

```
> db.placeProfiles.find({_id:70000}).pretty()
{
        "_id" : 70000,
        "acceptedPaymentModes" : "any",
        "address" : {
                "city" : "San Luis Potosi",
                "country" : "Mexico",
                "state" : "SLP",
                "street" : "Carretera Central Sn"
        },
        "cuisines" : "Mexican, Burgers",
        "parkingArragements" : "none",
        "placeFeatures" : {
                "accessibility" : "no_accessibility",
                "alcohol" : "No_Alcohol_Served",
                "area" : "closed",
                "dressCode" : "informal",
                "franchise" : "TRUE",
                "otherServices" : "Internet",
                "price" : "medium",
                "smokingArea" : "not permitted"
        },
        "placeName" : "Taco Jacks",
        "openingHour^" : [
                "Mon;Tue;Wed;Thu;Fri;09:00-20:00;",
                "Sat;12:00-18:00;",
                "Sun;12:00-18:00;"
        ]
}
>
```


2. MongoDb

Before:

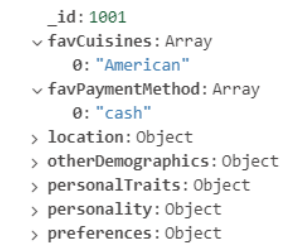


```
> db.userProfiles.find({_id:1108}).pretty()
{
        "_id" : 1108,
        "favCuisines" : "Cafe-Coffee_Shop, Sushi, Latin_American, Deli-Sandwiches, Mexican, Hot_Dogs, American, Fast_Food, Burgers, Asian, Pizzeria, Chinese,
Dessert-Ice_Cream, Cafeteria, Japanese, Game, Family, Seafood",
        "favPaymentMethod" : "VISA, cash, MasterCard-Eurocard",
        "location" : {
                "latitude" : 22.143524,
                "longitude" : -100.98756
        },
        "otherDemographics" : {
                "employment" : "student",
                "religion" : "Catholic"
        },
        "personalTraits" : {
                "birthYear" : 1983,
                "height" : 1.81,
                "maritalStatus" : "single",
                "weight" : 76
        },
        "personality" : {
                "drinkLevel" : "abstemious",
                "favColor" : "blue",
                "interest" : "technology",
                "typeOfWorker" : "thrifty-protector"
        },
        "preferences" : {
                "ambience" : "solitary",
                "budget" : "medium",
                "dressPreference" : "informal",
                "smoker" : false,
                "transport" : "public"
        }
}
```


I have decided to replace favCuisines and favPaymentMethod with the same name field just changing the type from comma separated string to array since we are allowed to modify appropriate databases. Here we are not adding any extra field just replacing same field just changing their data type from string to array to do certain operations and management in future.

```
db.userProfiles.aggregate(
   [
      { "$addFields":
          {
             "favCuisines": { "$split": [ "$favCuisines", ", " ] }
          }

      },
      { "$addFields":
          {
             "favPaymentMethod": { "$split": [ "$favPaymentMethod", ", " ] }
          }
```

```
    },
    {
    $merge:{
          into: 'userProfiles',
          on: '_id',
          whenMatched: 'merge',
          whenNotMatched: 'discard'
        }
    }
  ]
)
```



Here we are removing “Fast_Food” from favCuisines array and “cash” from favPaymentMethod array

```
db.userProfiles.update(
  { _id: 1108 },
  {
    $pull: { favCuisines: "Fast_Food",favPaymentMethod: "cash" }
  }
)
```

Then pushing “debit_cards” into favPaymentMethod array

```
db.userProfiles.update(
  { _id: 1108 },
  {
    $push: { favPaymentMethod: "debit_cards" }
  }
)
```

```
> db.userProfiles.update(
...     { _id: 1108 },
...     {
...       $pull: { favCuisines: "Fast_Food",favPaymentMethod: "cash" }
...     }
... )
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.userProfiles.update(
...     { _id: 1108 },
...     {
...       $push: { favPaymentMethod: "debit_cards" }
...     }
... )
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
>
```

```
_id: 1108
favCuisines: Array
    0: "Cafe-Coffee_Shop"
    1: "Sushi"
    2: "Latin_American"
    3: "Deli-Sandwiches"
    4: "Mexican"
    5: "Hot_Dogs"
    6: "American"
    7: "Burgers"
    8: "Asian"
    9: "Pizzeria"
    10: "Chinese"
    11: "Dessert-Ice_Cream"
    12: "Cafeteria"
    13: "Japanese"
    14: "Game"
    15: "Family"
    16: "Seafood"
favPaymentMethod: Array
    0: "VISA"
    1: "MasterCard-Eurocard"
    2: "debit_cards"
> location: Object
> otherDemographics: Object
> personalTraits: Object
> personality: Object
> preferences: Object
```

3. db.userProfiles.remove({_id:1063});

```
> db.userProfiles.remove({_id:1063})
WriteResult({ "nRemoved" : 1 })
```

After removing the user:

```
> db.userProfiles.find({_id:1063}).pretty()
>
```

4. Update user_ratings - IF EXISTS
    a. Before running the update query

```
cqlsh:fit5137a1_mrdb> SELECT user_fav_cuisines, user_fav_payment_method from user_ratings WHERE user_id=1108 AND rating_id=65;

 user_fav_cuisines
                                                                                        | user_fav_payment_method
-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------+----------------------------------------
 {'American', 'Asian', 'Burgers', 'Cafe-Coffee_Shop', 'Cafeteria', 'Chinese', 'Deli-Sandwiches', 'Dessert-Ice_Cream', 'Family', 'Fast_Food', 'Game', 'Hot_Dogs
', 'Japanese', 'Latin_American', 'Mexican', 'Pizzeria', 'Seafood', 'Sushi'} | {'MasterCard-Eurocard', 'VISA', 'cash'}

(1 rows)
```

UPDATE user_ratings SET user_fav_cuisines = user_fav_cuisines - {'Fast_Food'}, user_fav_payment_method = user_fav_payment_method - {'cash'}, user_fav_payment_method = user_fav_payment_method + {'debit_cards'} WHERE user_id= 1108 and rating_id IN(65, 66, 67, 68, 69, 70, 71, 72, 73, 74);

After doing the update operation we can check on rating id: 65

```
cqlsh:fit5137a1_mrdb> SELECT user_fav_cuisines, user_fav_payment_method from user_ratings WHERE user_id=1108 AND rating_id=65;

 user_fav_cuisines
                                                                          | user_fav_payment_method
-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------+----------------------------------------------
 {'American', 'Asian', 'Burgers', 'Cafe-Coffee_Shop', 'Cafeteria', 'Chinese', 'Deli-Sandwiches', 'Dessert-Ice_Cream', 'Family', 'Game', 'Hot_Dogs', 'Japanese'
, 'Latin_American', 'Mexican', 'Pizzeria', 'Seafood', 'Sushi'} | {'MasterCard-Eurocard', 'VISA', 'debit_cards'}

(1 rows)
cqlsh:fit5137a1_mrdb>
```

    b. Update user_id = 1063

DELETE FROM user_ratings WHERE user_id = 1063 AND rating_id = 137 IF EXISTS;
DELETE FROM user_ratings WHERE user_id = 1063 AND rating_id = 138 IF EXISTS;
DELETE FROM user_ratings WHERE user_id = 1063 AND rating_id = 139 IF EXISTS;
DELETE FROM user_ratings WHERE user_id = 1063 AND rating_id = 140 IF EXISTS;
DELETE FROM user_ratings WHERE user_id = 1063 AND rating_id = 141 IF EXISTS;

```
cqlsh:fit5137a1_mrdb> DELETE FROM user_ratings WHERE user_id = 1063 AND rating_id = 137 IF EXISTS;

 [applied]
-----------
      True

cqlsh:fit5137a1_mrdb> DELETE FROM user_ratings WHERE user_id = 1063 AND rating_id = 138 IF EXISTS;

 [applied]
-----------
      True

cqlsh:fit5137a1_mrdb> DELETE FROM user_ratings WHERE user_id = 1063 AND rating_id = 139 IF EXISTS;

 [applied]
-----------
      True

cqlsh:fit5137a1_mrdb> DELETE FROM user_ratings WHERE user_id = 1063 AND rating_id = 140 IF EXISTS;

 [applied]
-----------
      True

cqlsh:fit5137a1_mrdb> DELETE FROM user_ratings WHERE user_id = 1063 AND rating_id = 141 IF EXISTS;

 [applied]
-----------
      True
```

5. INSERT INTO user_ratings (rating_id,user_id,place_id,rating_place,rating_food,rating_service,user_personal_traits,user_personality, user_preferences, user_other_demographics, user_fav_cuisines,user_fav_payment_method) VALUES(7777, 1060, 70000, 2, 1, 2, {birth_year: 1991, weight: 82, height: 1.84, marital_status: 'single'}, {interest: 'technology', type_of_worker: 'thrifty-protector', fav_color: 'blue', drink_level: 'casual drinker'}, {budget: 'medium', smoker: FALSE, dress_preference: 'formal', ambience: 'family', transport: 'public'}, {religion: 'Catholic', employment: 'student'}, {'Burgers', 'Cafeteria', 'Pizzeria', 'Juice', 'American', 'Tex-Mex', 'Spanish', 'Mexican', 'Fast_Food', 'Cafe-Coffee_Shop', 'Soup', 'Hot_Dogs', 'Italian'}, {'cash'});

```
cqlsh:fit5137a1_mrdb> select * from user_ratings where user_id = 1060 and rating_id = 7777;

 user_id | rating_id | place_id | rating_food | rating_place | rating_service | user_fav_cuisines
                                                                               | user_fav_payment_method | user_other_demographics                              | user_
personal_traits                                                  | user_personality
                     | user_preferences
---------+-----------+----------+-------------+--------------+----------------+-----------------------------------------------------------------
-----------------------------------------------------------------+-------------------------+-------------------------------------------------------+------
------------------------------------------------------------------+--------------------------------------------------------------------------------
---------------------+------------------------------------------------------------------------------------------
    1060 |      7777 |    70000 |           1 |            2 |              2 | {'American', 'Burgers', 'Cafe-Coffee_Shop', 'Cafeteria', 'Fast_Food', 'Hot_Dog
s', 'Italian', 'Juice', 'Mexican', 'Pizzeria', 'Soup', 'Spanish', 'Tex-Mex'} |                {'cash'} | {religion: 'Catholic', employment: 'student'} | {birt
h_year: 1991, weight: 82, height: 1.84, marital_status: 'single'} | {interest: 'technology', type_of_worker: 'thrifty-protector', fav_color: 'blue', drink_lev
el: 'casual drinker'} | {budget: 'medium', smoker: False, dress_preference: 'formal', ambience: 'family', transport: 'public'}

(1 rows)
```

C3. Querying the Management

1. How many users are there in the database?
   Database: FIT5137A1MRDB

   db.userProfiles.find().count()

```
> db.userProfiles.find().count()
137
>
```

2. How many places are there in the database?

   Database: FIT5137A1MRDB
   db.placeProfiles.find().count()

```
> db.placeProfiles.find().count()
131
>
```

3. How many reviews were made in the database?

   SELECT COUNT(*) FROM place_ratings;

```
cqlsh:fit5137a1_mrdb> SELECT COUNT(*) FROM place_ratings;

 count
-------
  1161

(1 rows)
```

4. How many reviews were created by places having public parking arrangements?

I had to create index-**parking_arrangements** because here the primary key is already indexed not parking_arrangements. An index provides access data using attributes other than the partition key. Using indexing makes our query operation fast, efficient lookup of data matching a provided condition.

CREATE INDEX ON place_ratings (parking_arrangements);

SELECT COUNT (*) FROM place_ratings WHERE parking_arrangements = 'public';

```
cqlsh:fit5137a1_mrdb> CREATE INDEX ON place_ratings (parking_arrangements);
cqlsh:fit5137a1_mrdb>
```

```
cqlsh:fit5137a1_mrdb> SELECT COUNT (*) FROM place_ratings WHERE parking_arrangements = 'public';

 count
-------
   182

(1 rows)
```

5. Cassandra - Database FIT5137A1_MRDB

I have created an Index **user_personality.** because here the primary key is already indexed not parking_arrangements. An index provides access data using

attributes(**user_personality**) other than the partition key. Using indexing makes our query operation fast, efficient lookup of data matching a provided condition.

CREATE INDEX ON user_ratings (user_personality);

SELECT user_id, rating_place, user_personality FROM user_ratings where user_personality={interest: 'technology', type_of_worker: 'thrifty-protector', fav_color: 'blue', drink_level: 'casual drinker'};

user_id	rating_place	user_personality
1018	0	{interest: 'technology', type_of_worker: 'thrifty-protector', fav_color: 'blue', drink_level: 'casual drinker'}
1018	2	{interest: 'technology', type_of_worker: 'thrifty-protector', fav_color: 'blue', drink_level: 'casual drinker'}
1018	1	{interest: 'technology', type_of_worker: 'thrifty-protector', fav_color: 'blue', drink_level: 'casual drinker'}
1018	1	{interest: 'technology', type_of_worker: 'thrifty-protector', fav_color: 'blue', drink_level: 'casual drinker'}
1018	1	{interest: 'technology', type_of_worker: 'thrifty-protector', fav_color: 'blue', drink_level: 'casual drinker'}
1018	1	{interest: 'technology', type_of_worker: 'thrifty-protector', fav_color: 'blue', drink_level: 'casual drinker'}
1018	0	{interest: 'technology', type_of_worker: 'thrifty-protector', fav_color: 'blue', drink_level: 'casual drinker'}
1018	1	{interest: 'technology', type_of_worker: 'thrifty-protector', fav_color: 'blue', drink_level: 'casual drinker'}
1018	0	{interest: 'technology', type_of_worker: 'thrifty-protector', fav_color: 'blue', drink_level: 'casual drinker'}
1018	1	{interest: 'technology', type_of_worker: 'thrifty-protector', fav_color: 'blue', drink_level: 'casual drinker'}
1068	1	{interest: 'technology', type_of_worker: 'thrifty-protector', fav_color: 'blue', drink_level: 'casual drinker'}
1068	0	{interest: 'technology', type_of_worker: 'thrifty-protector', fav_color: 'blue', drink_level: 'casual drinker'}
1068	1	{interest: 'technology', type_of_worker: 'thrifty-protector', fav_color: 'blue', drink_level: 'casual drinker'}
1068	0	{interest: 'technology', type_of_worker: 'thrifty-protector', fav_color: 'blue', drink_level: 'casual drinker'}
1068	1	{interest: 'technology', type_of_worker: 'thrifty-protector', fav_color: 'blue', drink_level: 'casual drinker'}
1068	1	{interest: 'technology', type_of_worker: 'thrifty-protector', fav_color: 'blue', drink_level: 'casual drinker'}
1068	1	{interest: 'technology', type_of_worker: 'thrifty-protector', fav_color: 'blue', drink_level: 'casual drinker'}
1068	0	{interest: 'technology', type_of_worker: 'thrifty-protector', fav_color: 'blue', drink_level: 'casual drinker'}
1060	2	{interest: 'technology', type_of_worker: 'thrifty-protector', fav_color: 'blue', drink_level: 'casual drinker'}
1060	2	{interest: 'technology', type_of_worker: 'thrifty-protector', fav_color: 'blue', drink_level: 'casual drinker'}
1060	2	{interest: 'technology', type_of_worker: 'thrifty-protector', fav_color: 'blue', drink_level: 'casual drinker'}
1060	1	{interest: 'technology', type_of_worker: 'thrifty-protector', fav_color: 'blue', drink_level: 'casual drinker'}
1060	2	{interest: 'technology', type_of_worker: 'thrifty-protector', fav_color: 'blue', drink_level: 'casual drinker'}

6. What are the place ids and ratings for food for places serving only Pizzeria cuisine?

I have created an Index -**cuisines.** because here the primary key is already indexed not parking_arrangements. An index provides access data using attributes(**cuisines**) other than the partition key. Using indexing makes our query operation fast, efficient lookup of data matching a provided condition.

CREATE INDEX ON place_ratings (cuisines);

SELECT place_id, rating_food, cuisines FROM place_ratings where cuisines CONTAINS 'Pizzeria';

place_id	rating_food	cuisines
132869	1	{'Pizzeria'}
132869	2	{'Pizzeria'}
132869	1	{'Pizzeria'}
132869	2	{'Pizzeria'}
132869	0	{'Pizzeria'}
132869	2	{'Pizzeria'}
132733	2	{'Pizzeria'}
132733	1	{'Pizzeria'}
132733	1	{'Pizzeria'}
132733	1	{'Pizzeria'}
132733	1	{'Pizzeria'}
132733	1	{'Pizzeria'}
132733	0	{'Pizzeria'}
132733	2	{'Pizzeria'}
132733	2	{'Pizzeria'}
132733	2	{'Pizzeria'}
135058	2	{'Pizzeria'}
135058	1	{'Pizzeria'}
135058	0	{'Pizzeria'}
135058	1	{'Pizzeria'}
135058	2	{'Pizzeria'}
135058	2	{'Pizzeria'}
135058	1	{'Pizzeria'}
135058	0	{'Pizzeria'}
135058	1	{'Pizzeria'}
135058	2	{'Pizzeria'}
135058	1	{'Pizzeria'}
135058	1	{'Pizzeria'}
135058	2	{'Pizzeria'}
135058	0	{'Pizzeria'}
135058	0	{'Pizzeria'}
135058	0	{'Pizzeria'}
135058	0	{'Pizzeria'}
135058	2	{'Pizzeria'}

7. Display all users who are students and prefer a medium budget restaurant?

Database - FIT5137A1MRDB

```
db.userProfiles.aggregate([
{ $project : {_id : 0}},
{ $match : {"otherDemographics.employment":"student","preferences.budget":"medium"} }
]).pretty()
```


```
{
        "favCuisines" : [
                "Mexican"
        ],
        "favPaymentMethod" : [
                "cash"
        ],
        "location" : {
                "latitude" : 22.137178,
                "longitude" : -101.01317
        },
        "otherDemographics" : {
                "employment" : "student",
                "religion" : "Catholic"
        },
        "personalTraits" : {
                "birthYear" : 1990,
                "height" : 1.58,
                "maritalStatus" : "single",
                "weight" : 50
        },
        "personality" : {
                "drinkLevel" : "casual drinker",
                "favColor" : "yellow",
                "interest" : "technology",
                "typeOfWorker" : "thrifty-protector"
        },
        "preferences" : {
                "ambience" : "family",
                "budget" : "medium",
                "dressPreference" : "informal",
                "smoker" : false,
                "transport" : "public"
        }
}
```


8. Display all users who like Bakery cuisines and combine your output with all places having Bakery cuisines.

Here I have decided to change the data type cuisines and acceptedPaymentModes fields from string to array without adding any fields since we are allowed to bring required changes. Moreover, we can retrieve or add data more easily compared with comma separated strings.

```
db.placeProfiles.aggregate(
   [
      { "$addFields":
           {
              "cuisines": { "$split": [ "$cuisines", ", " ] }
           }

      },
      { "$addFields":
           {
              "acceptedPaymentModes": { "$split": [ "$acceptedPaymentModes", ", " ] }
           }

     },
     {
     $merge:{
           into: 'placeProfiles',
           on: '_id',
           whenMatched: 'merge',
           whenNotMatched: 'discard'
         }
     }
   ]
)
```

After running above query:


```
db.userProfiles.aggregate([

  {$match: {favCuisines:"Bakery"}},
  {
    $lookup:
      {
        from: "placeProfiles",
        pipeline: [
          {$match: {cuisines: "Bakery"}},
          { $project: { _id: 0, placeName: 1 } }
```

```
        ],
        as: "bakeryRestaurants"
      }
   }
]).pretty()
```

```
{
        "_id" : 1004,
        "favCuisines" : [
                "Bakery",
                "Breakfast-Brunch",
                "Japanese",
                "Contemporary",
                "Mexican",
                "Bagels",
                "Cafe-Coffee_Shop",
                "Continental-European",
                "Cafeteria"
        ],
        "favPaymentMethod" : [
                "cash",
                "bank_debit_cards"
        ],
        "location" : {
                "latitude" : 18.867,
                "longitude" : -99.183
        },
        "otherDemographics" : {
                "employment" : "professional",
                "religion" : "none"
        },
        "personalTraits" : {
                "birthYear" : 1940,
                "height" : 1.53,
                "maritalStatus" : "single",
                "weight" : 44
        },
        "personality" : {
                "drinkLevel" : "abstemious",
                "favColor" : "green",
                "interest" : "variety",
                "typeOfWorker" : "hard-worker"
        },
```



```
        "preferences" : {
                "ambience" : "family",
                "budget" : "medium",
                "dressPreference" : "informal",
                "smoker" : false,
                "transport" : "public"
        },
        "bakeryRestaurants" : [
                {
                        "placeName" : "Chaires"
                }
        ]
}
```

9. Display International restaurants that are open on Sunday.

```
db.placeProfiles.find(
  { openingHour : { $elemMatch : { "$regex": "^Sun.*" } }, "cuisines":"International"}).pretty()
```

```
{
        "_id" : 134986,
        "acceptedPaymentModes" : [
                "any"
        ],
        "address" : {
                "city" : "Cuernavaca",
                "country" : "Mexico",
                "state" : "Morelos",
                "street" : "Ricardo Linares 107"
        },
        "cuisines" : [
                "International"
        ],
        "location" : {
                "latitude" : 18.928798,
                "longitude" : -99.239513
        },
        "parkingArragements" : "yes",
        "placeFeatures" : {
                "accessibility" : "no_accessibility",
                "alcohol" : "Wine-Beer",
                "area" : "closed",
                "dressCode" : "formal",
                "franchise" : "f",
                "otherServices" : "none",
                "price" : "high",
                "smokingArea" : "none"
        },
        "placeName" : "Restaurant Las Mananitas",
        "openingHour" : [
                "Mon;Tue;Wed;Thu;Fri;00:00-23:30;",
                "Sat;00:00-23:30;",
                "Sun;00:00-23:30;"
        ]
}
```


10. Display the average place rating, average food rating, and average service rating for puesto de tacos. Show the average values in decimal points.

I have created an Index -**place_name.** Because here the primary key is already indexed not parking_arrangements. An index provides access data using attributes(**place_name**) other than the partition key. Using indexing makes our query operation fast, efficient lookup of data matching a provided condition.

CREATE INDEX ON place_ratings (place_name);

SELECT AVG(CAST(rating_place as decimal)) AS rating_place, AVG(CAST(rating_food as decimal)) AS rating_food, AVG(CAST(rating_service as decimal)) AS rating_service, place_name FROM place_ratings WHERE place_name='puesto de tacos' GROUP BY place_id;

```
 rating_place | rating_food | rating_service | place_name
--------------+-------------+----------------+-----------------
            2 |           2 |              1 | puesto de tacos

(1 rows)
```

11. Display the average age according to each drinker level

```
db.userProfiles.aggregate([
  {
  $group: {
    _id: {
     "drinkLevel":"$personality.drinkLevel"
    },

  averageAge: {
```

```
    $avg:

     {
       $divide: [{ $subtract: [ new Date(), new Date("$personalTraits.birthYear") ] },(365 *
24*60*60*1000)]
     }
  }

  }},
  {
    $project: {
       _id: 0,
       drinkLevel: "$_id.drinkLevel",
       averageAge: 1
    }
   },
   {$sort:{"drinkLevel":1}}
]).pretty()
```

```
{ "averageAge" : 50.79603560369102, "drinkLevel" : "abstemious" }
{ "averageAge" : 50.79603560369101, "drinkLevel" : "casual drinker" }
{ "averageAge" : 50.79603560369102, "drinkLevel" : "social drinker" }
>
```

12. For each user whose favourite cuisine is Family, display the place ID, the place rating, the food rating and the user's budget.

I have created an Index -**user_fav_cuisines.** Because here the primary key is already indexed not parking_arrangements. An index provides access data using attributes(**user_fav_cuisines**) other than the partition key. Using indexing makes our query operation fast, efficient lookup of data matching a provided condition.

CREATE INDEX ON user_ratings (user_fav_cuisines);

SELECT place_id, rating_place, rating_food, user_preferences.budget FROM user_ratings WHERE user_fav_cuisines CONTAINS 'Family';

place_id	rating_place	rating_food	user_preferences.budget
132754	1	2	medium
132862	2	1	medium
135060	1	1	medium
135059	2	1	medium
132834	2	2	medium
135058	1	1	medium
135038	2	2	medium
132825	1	1	medium
135052	2	1	medium
135063	0	0	medium
135079	1	2	medium
135066	0	0	low
132921	1	1	low
135052	1	1	low
132937	1	1	low
132951	1	1	low
132925	1	1	low
132872	1	1	low
132875	1	1	low
135085	1	1	low
135042	1	1	low

13. What are the top 3 most popular ambiences (friends/ family/ solitary) for a single when going to a Japanese restaurant?

```
db.userProfiles.aggregate([
 {
   $match: {"favCuisines": "Japanese" }
 },
 {
  $group: {
       _id: "$preferences.ambience",
       totalAmount: { $sum: 1 },
    }

 },
 {
   $project: {
     _id: 0,
     ambience: "$_id",
     totalAmount: 1
  }
},
{ $sort : { 'ambience' : 1,'totalAmount' : 1 } },
{ $limit : 3 }
]).pretty()
```

```
{ "totalAmount" : 3, "ambience" : "family" }
{ "totalAmount" : 3, "ambience" : "friends" }
{ "totalAmount" : 1, "ambience" : "solitary" }
>
```

14. List the names of unique cuisines in the database.

```
db.placeProfiles.aggregate([
{

    $unwind: "$cuisines"
},
{
    $group: {
          _id: {
                _id: "$_id"
          },
          distinctCuisines: {
                $addToSet: "$cuisines"
          }
    }
},
    {
```

```
    "$unwind": "$distinctCuisines"
    },
    {
      "$group": {
            _id: {
                  "distinctCuisines": "$distinctCuisines"
            }
      }
    },
    {
    $project: {
       _id: 0,
       cuisinesName: "$_id.distinctCuisines"
    }
   },
   {$match:{"cuisinesName":{$ne: "" }}}

]).pretty()
```

```
{ "cuisinesName" : "Seafood" }
{ "cuisinesName" : "Breakfast-Brunch" }
{ "cuisinesName" : "American" }
{ "cuisinesName" : "Bakery" }
{ "cuisinesName" : "Chinese" }
{ "cuisinesName" : "Cafeteria" }
{ "cuisinesName" : "Japanese" }
{ "cuisinesName" : "Game" }
{ "cuisinesName" : "Mediterranean" }
{ "cuisinesName" : "International" }
{ "cuisinesName" : "Italian" }
{ "cuisinesName" : "Contemporary" }
{ "cuisinesName" : "Mexican" }
{ "cuisinesName" : "Vietnamese" }
{ "cuisinesName" : "Bar_Pub_Brewery" }
{ "cuisinesName" : "Armenian" }
{ "cuisinesName" : "Cafe-Coffee_Shop" }
{ "cuisinesName" : "Pizzeria" }
{ "cuisinesName" : "Regional" }
{ "cuisinesName" : "Fast_Food" }
Type "it" for more
> it
{ "cuisinesName" : "Burgers" }
{ "cuisinesName" : "Family" }
{ "cuisinesName" : "Bar" }
>
```

15. Display all of the restaurants and indicate using a separate field/column whether the restaurant includes mexican cuisines

```
db.placeProfiles.aggregate(
  [
    {
      $project:
       {
         _id: 0,
         placeName: 1,
```

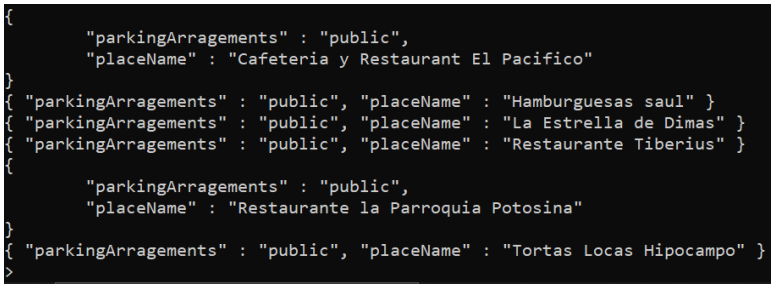
```
        includeMexicanCuisines:
          {
            $cond: { if: { $in: [ "Mexican", "$cuisines" ] }, then: "serves mexican food", else:
"doesn't serves mexican food" }
          }
        }
    },
      {$sort:{"_id":1}}
  ]
)
```

```
{ "placeName" : "puesto de gorditas", "includeMexicanCuisines" : "doesn't serves mexican food" }
{ "placeName" : "cafe ambar", "includeMexicanCuisines" : "doesn't serves mexican food" }
{ "placeName" : "churchs", "includeMexicanCuisines" : "doesn't serves mexican food" }
{ "placeName" : "Cafe Chaires", "includeMexicanCuisines" : "doesn't serves mexican food" }
{ "placeName" : "McDonalds Centro", "includeMexicanCuisines" : "doesn't serves mexican food" }
{ "placeName" : "Gorditas Dona Tota", "includeMexicanCuisines" : "serves mexican food" }
{ "placeName" : "tacos de barbacoa enfrente del Tec", "includeMexicanCuisines" : "serves mexican food" }
{ "placeName" : "Hamburguesas La perica", "includeMexicanCuisines" : "serves mexican food" }
{ "placeName" : "Pollo_Frito_Buenos_Aires", "includeMexicanCuisines" : "doesn't serves mexican food" }
{ "placeName" : "carnitas_mata", "includeMexicanCuisines" : "serves mexican food" }
{ "placeName" : "la perica hamburguesa", "includeMexicanCuisines" : "doesn't serves mexican food" }
{ "placeName" : "palomo tec", "includeMexicanCuisines" : "serves mexican food" }
{ "placeName" : "Carnitas Mata Calle 16 de Septiembre", "includeMexicanCuisines" : "doesn't serves mexican food" }
{ "placeName" : "carnitas mata calle Emilio Portes Gil", "includeMexicanCuisines" : "doesn't serves mexican food" }
{ "placeName" : "tacos abi", "includeMexicanCuisines" : "serves mexican food" }
{ "placeName" : "TACOS CORRECAMINOS", "includeMexicanCuisines" : "serves mexican food" }
{ "placeName" : "little pizza Emilio Portes Gil", "includeMexicanCuisines" : "doesn't serves mexican food" }
{ "placeName" : "TACOS EL GUERO", "includeMexicanCuisines" : "serves mexican food" }
{ "placeName" : "Gorditas Dona Tota", "includeMexicanCuisines" : "serves mexican food" }
{ "placeName" : "tacos de la estacion", "includeMexicanCuisines" : "serves mexican food" }
Type "it" for more
>
```

Extra five questions:

1. Display all of the restaurant's names in the city "San Luis Potosi" and their parking arrangement "public" and order by place name.

```
db.placeProfiles.aggregate(
  [
    {
      $match : {"address.city":"San Luis Potosi","parkingArragements":"public"}
    },
    {
      $project:
       {
        _id: 0,
        placeName: 1,
        parkingArragements: 1
       }
    },
      {$sort:{"placeName":1}}
  ]
).pretty()
```



2. List total number of user for each preferences ambience

```
db.userProfiles.aggregate(
  [
    {
      $project:
       {
        _id: 0,
        preferences:1
       }
    },

   {
     $group:
     {
      _id:{
      "ambience":"$preferences.ambience"
      },

      totalNoPersons: { $sum: 1 }
     }
   },
   {
    $project: {
      _id: 0,
      ambience: "$_id.ambience",
      totalNoPersons: 1
    }
   },
     {$sort:{"totalNoPersons":1}}
  ]
).pretty()
```

```
{ "totalNoPersons" : 4, "ambience" : "" }
{ "totalNoPersons" : 4, "ambience" : "public" }
{ "totalNoPersons" : 14, "ambience" : "solitary" }
{ "totalNoPersons" : 46, "ambience" : "friends" }
{ "totalNoPersons" : 69, "ambience" : "family" }
>
```

3. No of restaurants group by smokingArea features

```
db.placeProfiles.aggregate([
 {
   $group: {
     _id: {
      "smokingArea":"$placeFeatures.smokingArea"
     },
     totalAmount: { $sum: 1 },
    }
 },
 {
   $project: {
     _id: 0,
     smokingArea: "$_id.smokingArea",
     totalAmount: 1
    }
  },
  {$sort:{"totalAmount":-1}}
]).pretty()
```

```
{ "totalAmount" : 70, "smokingArea" : "none" }
{ "totalAmount" : 26, "smokingArea" : "not permitted" }
{ "totalAmount" : 24, "smokingArea" : "section" }
{ "totalAmount" : 9, "smokingArea" : "permitted" }
{ "totalAmount" : 2, "smokingArea" : "only at bar" }
>
```

4. List user_id, rating_place and user_personality of all the users other_demographics - religion: 'Catholic', employment: 'professional'

I have created an Index -**user_other_demographics.** Because here the primary key is already indexed not parking_arrangements. An index provides access data using attributes(**user_other_demographics**) other than the partition key. Using indexing makes our query operation fast, efficient lookup of data matching a provided condition.

```
CREATE INDEX ON user_ratings (user_other_demographics);
```

```
SELECT user_id, rating_place, user_personality FROM user_ratings where
user_other_demographics={religion: 'Catholic', employment: 'professional'};
```

user_id	rating_place	user_personality
1061	1	{interest: 'none', type_of_worker: 'hard-worker', fav_color: 'blue', drink_level: 'social drinker'}
1061	2	{interest: 'none', type_of_worker: 'hard-worker', fav_color: 'blue', drink_level: 'social drinker'}
1061	1	{interest: 'none', type_of_worker: 'hard-worker', fav_color: 'blue', drink_level: 'social drinker'}
1061	2	{interest: 'none', type_of_worker: 'hard-worker', fav_color: 'blue', drink_level: 'social drinker'}
1061	2	{interest: 'none', type_of_worker: 'hard-worker', fav_color: 'blue', drink_level: 'social drinker'}
1061	2	{interest: 'none', type_of_worker: 'hard-worker', fav_color: 'blue', drink_level: 'social drinker'}
1061	2	{interest: 'none', type_of_worker: 'hard-worker', fav_color: 'blue', drink_level: 'social drinker'}
1061	2	{interest: 'none', type_of_worker: 'hard-worker', fav_color: 'blue', drink_level: 'social drinker'}
1061	1	{interest: 'none', type_of_worker: 'hard-worker', fav_color: 'blue', drink_level: 'social drinker'}
1061	1	{interest: 'none', type_of_worker: 'hard-worker', fav_color: 'blue', drink_level: 'social drinker'}
1061	1	{interest: 'none', type_of_worker: 'hard-worker', fav_color: 'blue', drink_level: 'social drinker'}
1061	1	{interest: 'none', type_of_worker: 'hard-worker', fav_color: 'blue', drink_level: 'social drinker'}
1061	1	{interest: 'none', type_of_worker: 'hard-worker', fav_color: 'blue', drink_level: 'social drinker'}
1061	2	{interest: 'none', type_of_worker: 'hard-worker', fav_color: 'blue', drink_level: 'social drinker'}
1061	2	{interest: 'none', type_of_worker: 'hard-worker', fav_color: 'blue', drink_level: 'social drinker'}
1061	2	{interest: 'none', type_of_worker: 'hard-worker', fav_color: 'blue', drink_level: 'social drinker'}
1061	2	{interest: 'none', type_of_worker: 'hard-worker', fav_color: 'blue', drink_level: 'social drinker'}
1061	2	{interest: 'none', type_of_worker: 'hard-worker', fav_color: 'blue', drink_level: 'social drinker'}
1078	1	{interest: 'variety', type_of_worker: 'hard-worker', fav_color: 'blue', drink_level: 'social drinker'}
1078	1	{interest: 'variety', type_of_worker: 'hard-worker', fav_color: 'blue', drink_level: 'social drinker'}
1078	2	{interest: 'variety', type_of_worker: 'hard-worker', fav_color: 'blue', drink_level: 'social drinker'}
1078	2	{interest: 'variety', type_of_worker: 'hard-worker', fav_color: 'blue', drink_level: 'social drinker'}
1078	2	{interest: 'variety', type_of_worker: 'hard-worker', fav_color: 'blue', drink_level: 'social drinker'}
1078	0	{interest: 'variety', type_of_worker: 'hard-worker', fav_color: 'blue', drink_level: 'social drinker'}
1078	2	{interest: 'variety', type_of_worker: 'hard-worker', fav_color: 'blue', drink_level: 'social drinker'}
1117	1	{interest: 'technology', type_of_worker: 'thrifty-protector', fav_color: 'blue', drink_level: 'abstemious'}
1117	0	{interest: 'technology', type_of_worker: 'thrifty-protector', fav_color: 'blue', drink_level: 'abstemious'}
1117	1	{interest: 'technology', type_of_worker: 'thrifty-protector', fav_color: 'blue', drink_level: 'abstemious'}
1117	2	{interest: 'technology', type_of_worker: 'thrifty-protector', fav_color: 'blue', drink_level: 'abstemious'}

5. . Display all users who like to smoke and combine your output with all places not having smoking areas.

```
db.userProfiles.aggregate([

  {$match: {"preferences.smoker":true}},
  {
    $lookup:
      {
        from: "placeProfiles",
        pipeline: [
          {$match: {"placeFeatures.smokingArea": "not permitted"}},
          { $project: { _id: 0, placeName: 1 } }
        ],
        as: "smokingNotPermitted"
      }
  }
]).pretty()
```

```
            "VISA"
        ],
        "location" : {
                "latitude" : 18.871654,
                "longitude" : -99.183251
        },
        "otherDemographics" : {
                "employment" : "student",
                "religion" : "Catholic"
        },
        "personalTraits" : {
                "birthYear" : 1990,
                "height" : 1.67,
                "maritalStatus" : "single",
                "weight" : 80
        },
        "personality" : {
                "drinkLevel" : "casual drinker",
                "favColor" : "blue",
                "interest" : "variety",
                "typeOfWorker" : "thrifty-protector"
        },
        "preferences" : {
                "ambience" : "family",
                "budget" : "medium",
                "dressPreference" : "no preference",
                "smoker" : true,
                "transport" : "car owner"
        },
        "smokingNotPermitted" : [
                {
                        "placeName" : "Cafe Chaires"
                },
                {
                        "placeName" : "McDonalds Centro"
                },
                {
                        "placeName" : "Gorditas Dona Tota"
                },
                {
```


C.4. Summary Reports:

1. The mechanism of the Relational Database, Document Oriented Database, and Column-Oriented Database:

Relational Database: it is a widely used type of database, represented as a structured query language (SQL) type of databases. It has been developed based on a theory called the relational model of data that was founded by computer scientists whose name is E.F.Codd (Codd, 1970). In a relational database, the data is stored in predefined tables that are connected by key as a means of linking the tables together through the concept of the primary key and foreign key. Each table is formed into columns and rows, which will later be fed either a single set of data or multiple. The single object is stored in a complete row. Each table has a key that uniquely identifies each row called a primary key. Also, there is a key included from the related tables as a way of linking them. That is why it is called a relational database because each table has a relation with other tables.

Document-Oriented Database: it is categorized as non- non-structured query language (SQL), meaning it does not follow the same structure as a relational database. It does not require a pre-defined structure as the relational database. The document is the basic component of data, and it is comparable to rows in a relational database. The document consists of key-value pairs. Then we have something called a collection where all the documents are stored. The collection can hold an uncountable number of documents. It serves as a table in a relational database.

Column-Oriented Database: It is similar to a relational database. However, it stores the data tables in columnar format instead of row format. It gathers data in key-value pairs. The column family consists of logically grouped columns same as the table in the relational database. Moreover, the values in the column family are assigned row identifiers the same as the primary key in the relational database context. Also, one row can hold several column families.

2. The strengths and weaknesses of the Relational Database, Document Oriented Database, and Column-Oriented Database:

Database Type	Strength	Weakness
Relational database	**Data accuracy:** Relational databases provide a very high level of accuracy of data because of the concept of the primary key and foreign key. **Data Integrity:** certain functionalities such as cardinality and validation ensure the data are in the right table. **High Security:** specifying user privilege is achievable as the data is spread out over multiple tables. **Data redundancy:** The process of normalizing the data ensures reduction of data replication or either	**Unscalable data model:** difficulty modifying the data model because the data model must be pre – structure before implementing the database. Otherwise, complexity and impossibility will emerge during the modification. **Cost:** Every type of database has a certain cost, but a relational database is not affordable to maintain. For example, the purchasing cost of the software license and hiring database professionals such as data administrator.

	the complete elimination of data replication.	**The disappearance of data:** There is a high chance of losing the data because of the large number of tables. This is true especially when transferring the data to a new system.
Document oriented database	**Simplicity**: complex database design can significantly be simplified using a document-oriented database. For example, three tables with five relationships instead of 15 tables with many relationships using a relational database. **Easy retrieval of data:** all the information is stored in one document, so there is no need for a complex operation such as joins. **Flexibility**: The Document oriented database does not require the users to predefine the schema for the database. For example, some documents have the field of payment method where other documents simply do not.	**Transaction errors:** the absence of strict control due to the lack of compliance with the principle of ACID. **Integrity issues:** data integrity does not feature in a document-oriented database; it requires to be manually programmed**,** unlike a relational database. **Security issues**: several issues have been reported with a document-oriented database such as unauthorized access, unidentified deletion, and update due to malware infection (Kaur, Kaur, 2013)
Column oriented database	**High performance:** it is a powerful way to increase the speed of data retrieval and perform highly aggregated analysis. **Efficient storage:** dealing with a null value in the column more effectively. Also, the data block in the column-oriented database can hold more data than a row-based database. **Great for big data:** Column-oriented database is excellent for big data, especially when running queries because the database will scan the desired data instead of scanning the whole data.	**Limitation on retrieval:** difficulty when retrieving the entire row. **Complex Join request:** The advantage of high performance may be reduced when performing a complex join operation as the whole column must be scanned. **Inefficiency in delete and update operation:** when modifying a row in a column-oriented database, it is required to allocate the places on the disk, which sometimes becomes a tedious operation.

3. Real world examples of the application of the Relational Database, Document Oriented Database, and Column-Oriented Database:

Relational database:

There are many applications of a relational database in the real world. The banking industry uses many types of databases. However, a relational database is used when performing transactional activities such as withdrawal and deposit. The relational database is the most suitable for recording this sensitive information as they follow the concept of ACID. **Document oriented database:**

With the rise of big data, many organizations are becoming tempted to use a document-oriented database because it offers great flexibility when storing a huge amount of data and no predefined schema is necessary. In India, the organization of Aka Aadhar has built the biggest biometrics database that stores data about 1.2 billion residents. Aka Aadhar used a well-known type of document-oriented database called MongoDB (Gopinath, 2019).

Column oriented database:

With the emergence and evolution of data warehousing and business intelligence, developers have encountered limitations in a row-based database. This is especially true when analyzing a huge dataset in the data warehouse. It has been reported that a column-oriented database has gained popularity in the field of data warehousing and business intelligence, and this popularity has resulted in the increase of the application of the column-oriented database (Rabuzin, & Modrusan, 2014).

4. Database selection:

MonR has informed us of a sudden cut down on their budget, which resulted in the decision to drop down one of the databases either MongoDB or Cassandra. This was a sudden change, especially when almost 80% of our job is done. We needed to undertake extensive research, planning, and consultation to make the right decision. The CAP theorem dimension was utilized as the basis for our decision-making process. The CAP theorem dimension states that it is undoable to achieve the performance standards, which is availability, consistency, and partition-tolerance at once. However, a compromise must be made between the performance standards

(Abadi, 2012).

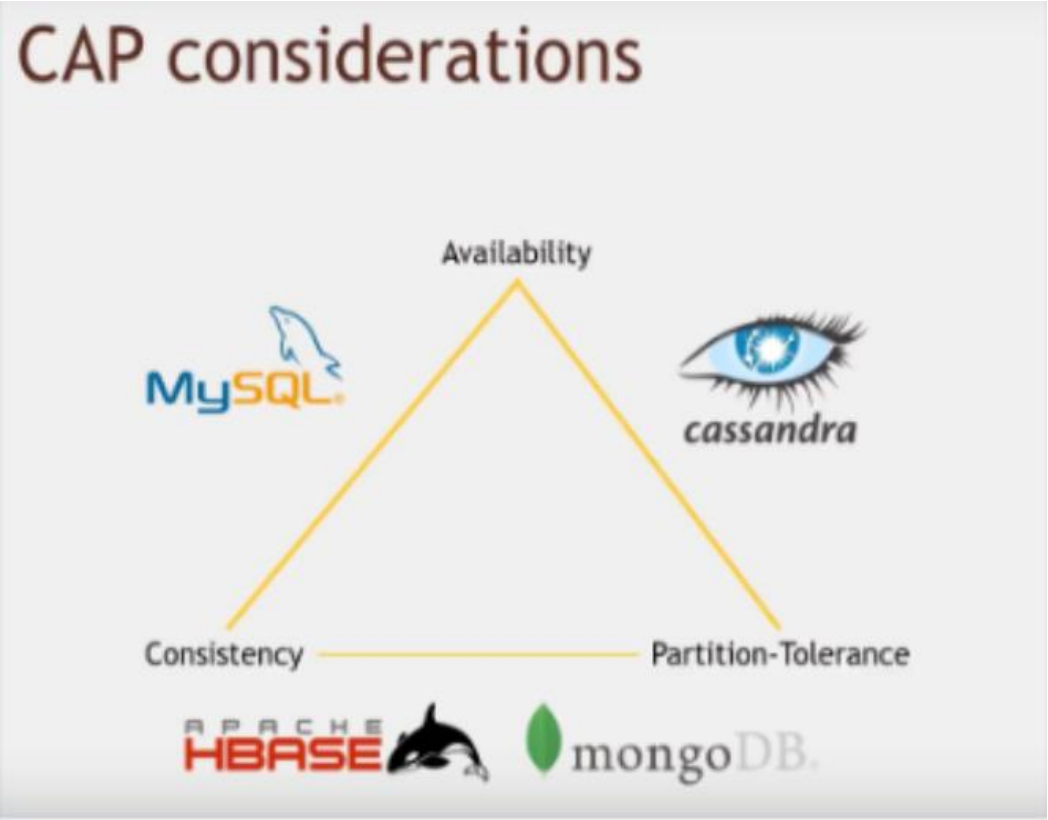


Illustration of CAP theorem dimension

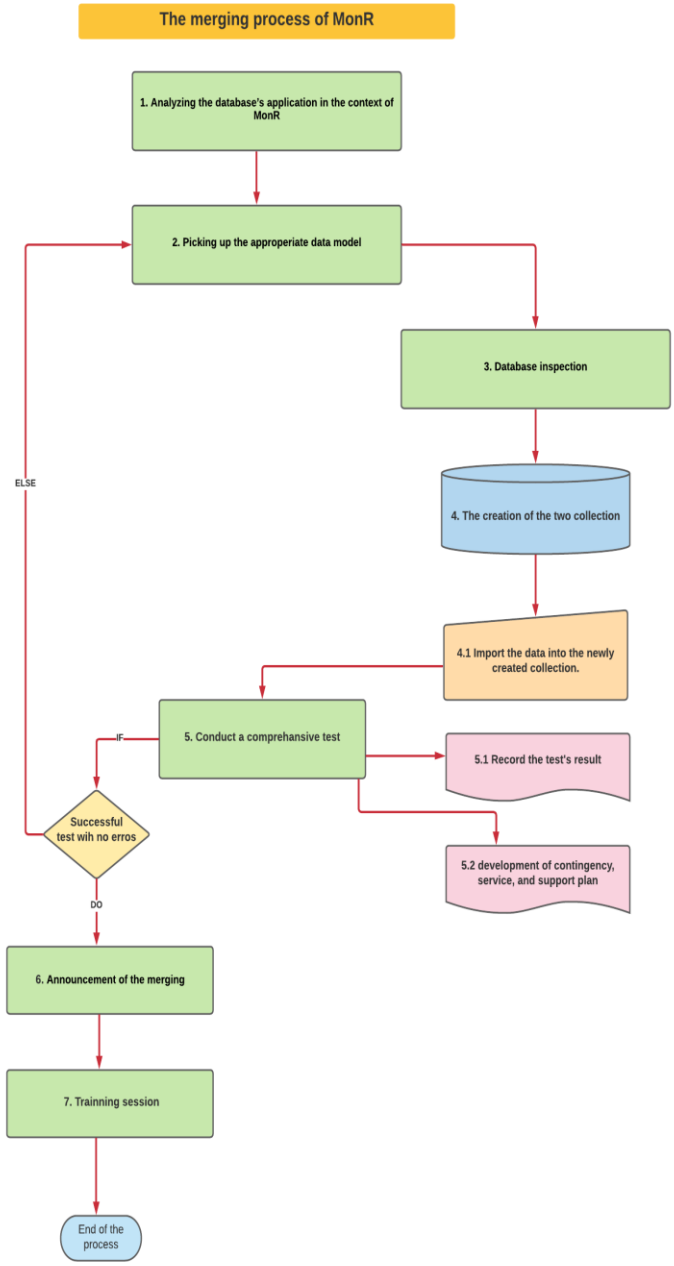
We decided that MongoDB would be the best fit in the context of MonR. We decided after evaluating the case in many vital aspects and looking for both features and functionalities of the databases.

5. The stages of the merging process + flow chart of the merging process:

The implementation of the merging process is a crucial one and requires a pre-established plan to ensure its success. A flowchart was constructed for the purpose of demonstrating the stages of how the merging process will be implemented. The table below is an explanation of each stage of the merging process.

Stage Number	Explanation
First	Detailed analysis of the database's application in the context of MonR. Specifying the scope of a database application to support the requirements of the use case.
Second	Deciding the most suitable data model based on the situation of MonR. The Reference model is ideal to be applied here as the model will provide fast writing

	operation into the newly created collection. In contract, the embedding model will be able to get the job done, but there will be more complications, especially in writing time and modification.
Third	Inspection of the structure and type of data of the Cassandra database to facilitate the merging process includes the conversion of the data type. Also, wrangling some data as MongoDB has no primary key like Cassandra to guarantee the consistency and quality of data in the database.
Fourth	Create two collections of user ratings and place ratings.
Fifth	Conduct a comprehensive test to evaluate whether all the pre-defined planning and requirements have been met.
Sixth	Announcement of the completion of the database merging
Seventh	Allocating training sessions to the employees to familiarize them with using the MongoDB database and exploring the features of MongoDB.



6. Database selection reasoning:

We picked MongoDB over Cassandra for the following reason:

- MongoDB will offer a great boost in achieving the consistency of data by allowing a fast data integration as it stores the data in JSON documents instead of tables in the relational databases.
- MongoDB has an unstructured data model, which means flexibility more than Cassandra. The flexibility of the data model enables us to modify the database only in the frontend of the database without changing the backend of the database.
- MongoDB has a power aggregation mechanism, which is a feature that surely, we do not want to give, especially during the analyzing the dataset.
- MongoDB has a graphical interface called MongoDB compass. It will assist us when examining the data, and performing operations such as filtering,

projecting, and aggregating the data. Also, it will be user-friendly for non – IT people.

- Cassandra would surely increase the cost in terms of money, efforts, and time because Cassandra has a structured schema, which means the re-design the whole database.
- Cassandra would be an ideal database if we are dealing with a huge database (big data) throughout several nodes. However, we are not dealing with huge amounts of data in this case.
- In case of clustering is needed, MongoDB is more superior as it can serve several configuration servers where Cassandra has none.
- We want to have a rich data model in order to be prepared if MonR wants to add more data into the database. This is because they may start analyzing data from Mexican stores in Australia or any other places.

Reference

Cod&, E. F. (1970). A Relational Model for Large Shared Data Banks. *Comm. ACM*, *13*(6), 37.

Kaur, H., & Kaur, K. J. (2013). A Review: Study of Document Oriented Databases and TheirSecurity. International Journal of Advanced Research in Computer Science, 4(8).

Sudhaa Gopinath. (2019, May 22). Real World Use Cases of MongoDB. Edureka. https://www.edureka.co/blog/real-world-use-cases-of-mongodb/

Rabuzin, K., & Modrusan, N. (2014). Business Intelligence and Column-Oriented Databases.

Abadi, D. (2012). Consistency tradeoffs in modern distributed database system design: CAP
is only part of the story. Computer, 45(2), 37-42.

C5: Connecting to Drivers

Steps to connect the MongoDB drivers:

1. Pip was not working in my existing python version 2.7. I had to install anaconda and python version 3.8.3 was used to run the script
2. I had to import following modules:

```
try:
    import pymongo
    from pymongo import MongoClient
    import pprint
    import pandas as pd
    import datetime
except Exception as e:
    print("We have missed some modules")
```

3. In the constructor of MongoClientDB class we connected to the localhost where MongoDB server is running and along with that I had to initialize the database and collection here. Whenever an object is created of MongoClientDB class a new connection is created along with the database and other three collections.

```
def __init__(self, dBName=None, collectionUser=None, collectionPlaces=None, collectionOpeningHrs=None):
    self.dBName = dBName
    self.collectionUser = collectionUser
    self.collectionPlaces = collectionPlaces
    self.collectionOpeningHrs = collectionOpeningHrs

    self.client = MongoClient("localhost", 27017, maxPoolSize=50)
    self.DB = self.client[self.dBName]
    self.collectionUser = self.DB[self.collectionUser]
    self.collectionPlaces = self.DB[self.collectionPlaces]
    self.collectionOpeningHrs = self.DB[self.collectionOpeningHrs]
    if self.collectionUser.drop():
        print("Collection dropped")
    if self.collectionPlaces.drop():
        print("profilePlaces dropped")
    if self.collectionOpeningHrs.drop():
         print("Opening hours dropped")
```

Rest of the code has been given in the python script file.

Steps to connect the Cassandra drivers:

1. Pip was not working in my existing python version 2.7. I had to install anaconda and python version 3.8.3 was used to run the script
2. Cassandra driver was installed using the following command:
    a. conda install -c conda-forge cassandra-driver
3. In a separate command prompt/terminal window had to run the cassandra server using
    a. Cassandra
4. We had to import only the cluster module in the cassandra driver using
    a.

```
try:
    from cassandra.cluster import Cluster
except Exception as e:
    print("We have missed some modules")
```

5. In order to connect to the cluster, we first created a cluster object using the cluster command and then pass in the expected list of hosts
    a.
```
cluster = Cluster(['localhost'])
```
6. Next we had to create a session to connect to the newly created cluster using
    a.
```
session = cluster.connect()
```
7. Rest of the code has been given to the python script.