**REDIS DATA-STRUCTURES USED:**

1. **For Storing Restaurants:**
   1. **HASHES** - to Store the Restaurant and their attributes like services, facilities, working days and hours.

KEY: restaurant:restID

EG:- restaurant:153

VALUE: {

name: 'tacos los volcanes',

address: 'Francisco I. Madero 145 Centro',

city: 'san luis potosi',

country: 'Mexico',

restID: '56',

state: 'san luis potos',

zip: '78290',

facilities\_ambience: 'solitary',

facilities\_parkingSpace: 'true',

facilities\_seatingArea: 'open',

services\_alcohol: 'false',

services\_smoking: 'true',

workingDays: 'Monday',

payments: 'cash',

cuisine: 'Mexican,Chinese,American,Indonesian',

priceRangeMin: '17',

priceRangeMax: '73',

openHours: '1:40 AM',

closeHours: '10:27 PM',

dresscode: 'informal'

}

* 1. **LISTS - Store** all the restaurant ids

**EG**:- restids [ 1, 2, 3 ……. ]

Lists are used instead of sets or sorted sets so functions like LRANGE can be performed as SETS output random values

* 1. **SETS   
       
       
     1.3.1 Store all the different kind of cuisines**

KEY: cuisines

VALUE: American, Indian, Chinese, Mexican, Cuban ……. etc.

**1.3.2** Store restids of all restaurants of a particular type of cuisine so we can query the restaurant by cuisine.

KEY: cuisine:cuisineName

EG:- cuisine:American

VALUE: 445

1. **Caching data**

For both Rating and Customer, we use a HASH as this was the easiest to implement from a mongo database that uses json.

Redis hashes are sets of keys with a string value. They are used for saving objects. A hash in Redis can store up to 2^32 – 1 key/value pairs. To set a hash in Redis we use the hSet Function with the key as the first parameter and the Json object as the second parameter. When we encounter a nested object in our data, we json.stringify the data so it fits into a single Redis HASH. Retrieving the hash is done with the hGetAll function. This function gets all key/values of the HASH. It returns an empty object if no key is found. When retrieving the object, we JSON.parse the nested objects again so we can use them in the frontend.

* 1. **Cache customer**

Key:customer:customerId

example key: customer:46

value:

{

"ambience": "family",

"budget": "low",

"customerID": 282,

"drinkLevel": "casual drinker",

"name": "Alix Bowering",

"smoker": false,

"dressCode": "informal",

"cuisine": ["Japanese","Malaysian","American"],

"paymentMethods": ["American\_Express"]

}

Nested json objects are JSON.stringify so they fit into a string variable.

* 1. **Cache Rating**

Key: rating:ratingId

Example key: rating:61996621181603ed94618714

value:

{

"Food": 5,

"Service": 4,

"cost": 4,

"overall": 4.2,

"parking": 4,

"ratingId": 1,

"restID": 160,

"waiting": 4,

"customer {"ambience":"family","budget":"medium","customerID":46,"drinkLevel":"abstemious","name":"Agatha Kinzett","smoker":false,"dressCode":"informal","cuisine":["Japanese","Malaysian","American"],"paymentMethods":["American\_Express"]}

}

Nested json objects are JSON.stringify so they fit into a string variable.