Functional Dependencies:

Chefmoz:

placeID → Rpayment, Rcuisine, hours, days, parking_lot

Geoplaces2:

placeID → latitude, longitude, name, city, state, address, country, fax, zip, alcohol, smoking_area, dress_code, accessibility, price, url, Rambience, franchise, area, other_services, the_geom_meter, name

Rating_final:

userID, placeID → rating, food_rating, service_rating

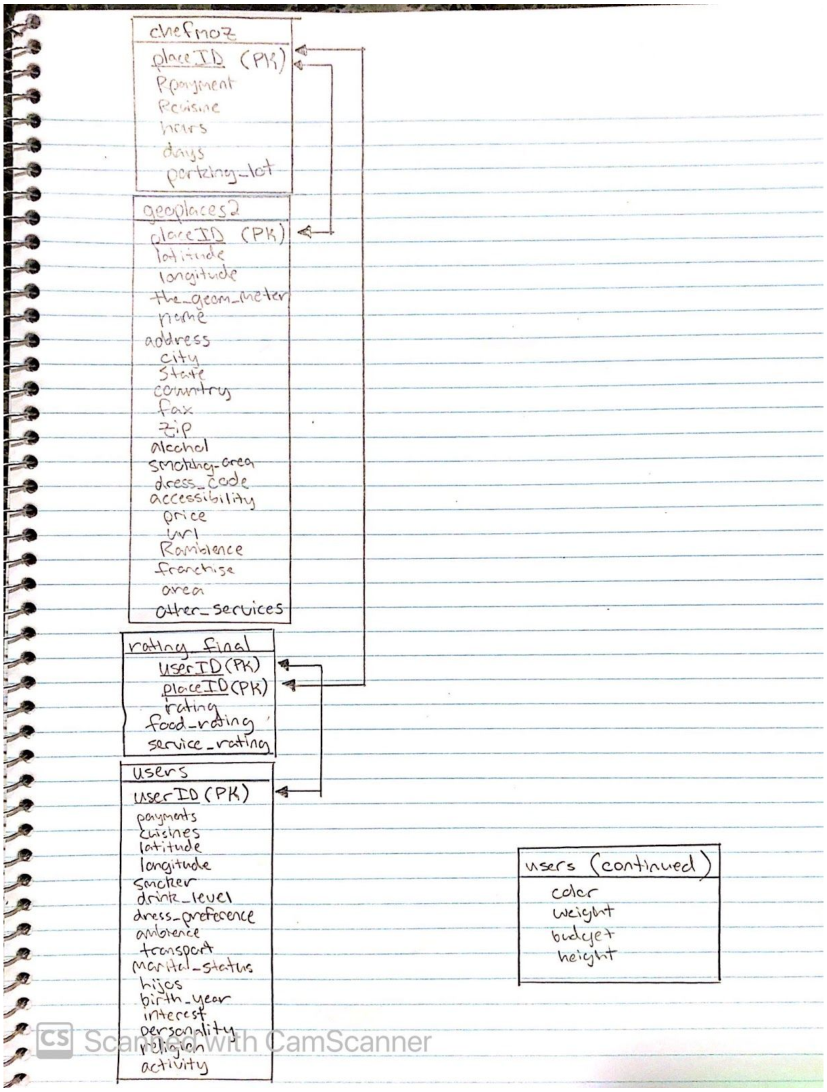
Users:

userID → payments, cuisines, latitude, longitude, smoker, drink_level, dress_preference, ambience, transport, marital_status, hijos, birth_year, interest, personality, religion, activity, color, weight, budget, height

Schema Design Notes:

1) Geoplaces2 could have been included in chefmoz, however since most of our queries use geoplaces2 or chefmoz separately we felt it would be smarter to separate the two relations since geoplaces2 is quite large. This way our queries are simpler.
2) We made payments and cuisines in the users table into arrays. We did this so that users could be in 3NF and we felt it was a logical way to store the data and was easily accessible.

Schema:



Create table statements:

```
// Below are two views we defined when creating our tables
CREATE VIEW cuisines AS
SELECT "userID", array_agg("Rcuisine") as cuisines
FROM usercuisine U
GROUP BY "userID";

CREATE VIEW payments AS
SELECT "userID", array_agg("Upayment") as payments
FROM userpayment U
GROUP BY "userID";

// Creating relation users
CREATE TABLE users AS
SELECT R."userID", payments, cuisines, latitude, longitude,
     smoker, drink_level, dress_preference, ambience,
     transport, marital_status,hijos, birth_year, interest,
     personality, religion, activity, color, weight, budget, height
   FROM userprofile R
   INNER JOIN payments P on R."userID" = P."userID"
   INNER JOIN cuisines C on P."userID" = C."userID"


// Creating relation chefmoz
CREATE TABLE chefmoz
AS (SELECT A."placeID", "Rpayment", "Rcuisine", hours, days, parking_lot
  FROM chefmozaccepts A
  INNER JOIN chefmozcuisine C ON A."placeID" = C."placeID"
  INNER JOIN chefmozhours4 H ON A."placeID" = H."placeID"
  INNER JOIN chefmozparking P ON A."placeID" = P."placeID")
```

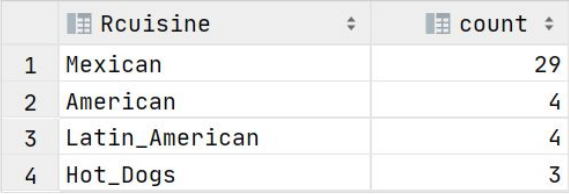
Then the other two tables are geoplaces2 and rating_final. They are already defined from the imported csv.

Questions:

a) Logic: To determine the most popular cuisine amongst those interested in Technology, we first filter the user table to create a view that contains only users where their interest attribute is technology. Next, we perform a group by on attribute on U.Rcuisine, sorted in descending order, so that the first value will be the most popular cuisine.

   Query 1 for most popular cuisine (for Tech interest):

```
SELECT U.”Rcuisine”, count(distinct U.”userID”)
FROM “user” U
WHERE U.”interest” = ‘technology’
GROUP BY U.”Rcuisine”
ORDER BY 2 DESC
```



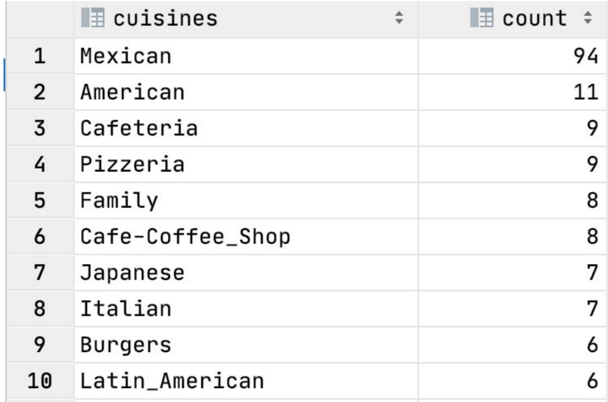
	Rcuisine	count
1	Mexican	29
2	American	4
3	Latin_American	4
4	Hot_Dogs	3

b) Logic: In order to find the cuisine that is most popular and least popular we decided to concatenate columns Upayment, smoker, drink_level, dress_preference, ambience, transport, marital_status, hijos, interest, personality, religion, activity, color, budget from the table user, as defined above. By doing this we are able to have a new column which includes different types of users that favor a certain cuisine. By selecting only Rcuisine and our new column we are able to group by Rcuisine and retrieve the count of distinct entries in the new column for each group. Then we can sort this group on the count column and get the most popular cuisine by sorting by descending and the least popular cuisine by sorting by ascending. In the end we get a column of different_users which is a numerical entry for how many different types of users a cuisine is preferred by.

Query 1 for most popular:

```
SELECT cuisines, count(*)
FROM (SELECT unnest(cuisines) AS cuisines, concat_ws('-', "payments",
    "cuisines", smoker, drink_level, dress_preference, ambience,
    transport, marital_status, hijos, birth_year, interest,
    personality, religion, activity,
        color, budget)
    FROM "users" U) AS K
GROUP BY cuisines
ORDER BY 2 DESC
```

Answer 1 for most popular:

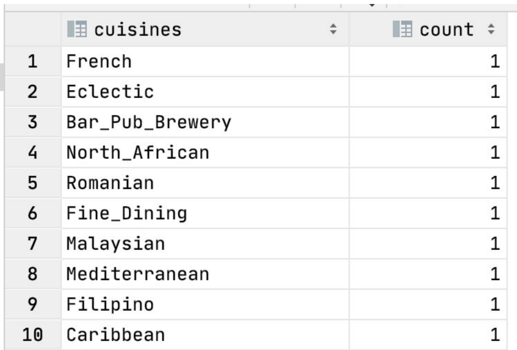


	cuisines	count
1	Mexican	94
2	American	11
3	Cafeteria	9
4	Pizzeria	9
5	Family	8
6	Cafe-Coffee_Shop	8
7	Japanese	7
8	Italian	7
9	Burgers	6
10	Latin_American	6

Query 2 for least popular:

```
SELECT cuisines, count(*)
FROM (SELECT unnest(cuisines) AS cuisines, concat_ws('-', "payments",
    "cuisines", smoker, drink_level, dress_preference, ambience,
    transport, marital_status, hijos, birth_year, interest,
    personality, religion, activity,
        color, budget)
    FROM "users" U) AS K
GROUP BY cuisines
ORDER BY 2 ASC
```

Answer 1 for least popular:



	cuisines	count
1	French	1
2	Eclectic	1
3	Bar_Pub_Brewery	1
4	North_African	1
5	Romanian	1
6	Fine_Dining	1
7	Malaysian	1
8	Mediterranean	1
9	Filipino	1
10	Caribbean	1

c) Logic: In order to recommend users valid restaurants, we create one more table labeled ‘Searching’ by joining rating_final, chefmoz, and geoplaces- We do this in order to add each restaurant’s opening times, closing times, average rating, days open, and type of cuisine served. By creating this table our following queries on the individual users are made much easier.

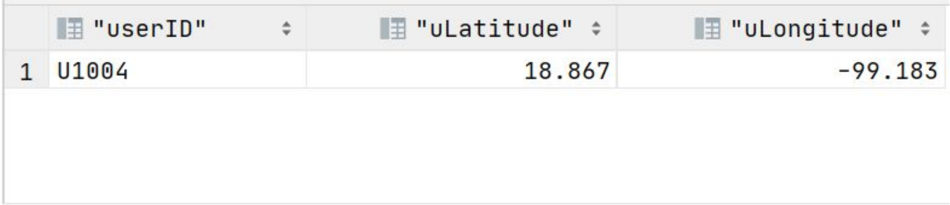
```
CREATE TABLE searching AS (
SELECT  DISTINCT C."placeID", g.name, g.latitude, g.longitude, g.price, C."Rcuisine", LEFT(C."hours", 5) AS
startTime, LEFT(RIGHT(C."hours", 6), 5) AS endTime, C."days", K."average"
FROM chefmoz C
INNER JOIN (SELECT "placeID", AVG(("rating" + "food_rating" + "service_rating") / 3) AS "average"
FROM rating_final
GROUP BY "placeID") K
ON C."placeID" = K."placeID"
INNER JOIN geoplaces2 g on C."placeID" = g."placeID");

UPDATE searching
SET endTime='24:00'
WHERE endTime='00:00';

// we used a Table here in order to edit the ‘00:00’ to a ‘24:00’ for ease of checking if the restaurant was open
```

The first customer has a preference for Mexican Food, Cares about Price, and prefers fast delivery/travel time. To create the query we first found the user’s latitude and longitude (and if generating a longer-term query that could function for any user we would replace the first query and magic numbers generated therein with a subquery to find that specific user’s longitude and latitude, but as we are only using it once this will serve).

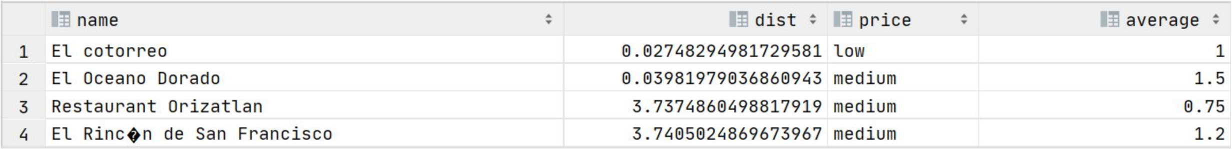
```
SELECT "userID", "latitude" AS "uLatitude", "longitude" AS "uLongitude"
FROM "user"
WHERE "userID"='U1004'
LIMIT 1;
```



	"userID"	"uLatitude"	"uLongitude"
1	U1004	18.867	-99.183

Now that we have the user’s location, we can create a distance variable between them and the restaurant locations. Using this we can now filter the restaurants by the user’s preferences, i.e. Monday at 18:00, Mexican Food, and sort them by the distance, then price, then average as per the user’s choice.

```
SELECT S."name", SQRT(POWER(S.latitude - 18.867, 2) + POWER(S.longitude + 99.183, 2)) AS "dist",
S."price", S."average"
FROM searching S
WHERE days LIKE '%' || 'Mon' || '%' AND endTime >= '18:00' AND startTime <= '18:00' AND "Rcuisine"='Mexican'
ORDER BY "dist", "price", "average" DESC;
```

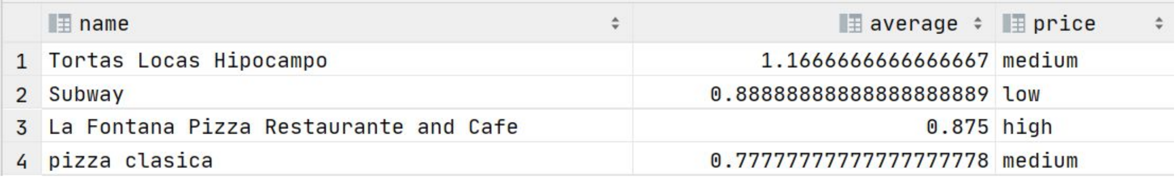


	name	dist	price	average
1	El cotorreo	0.02748294981729581	low	1
2	El Oceano Dorado	0.03981979036860943	medium	1.5
3	Restaurant Orizatlan	3.7374860498817919	medium	0.75
4	El Rinc�n de San Francisco	3.7405024869673967	medium	1.2

By looking at this query we can recommend the top three restaurants (El cotorreo, El Oceano Dorado, and Restaurant Orizatlan) to the first user. A deeper look at the query output reveals that the ‘price’ attribute is medium for most restaurants, with only 2 ‘low’ and 1 ‘high’ value. With this in mind, we will recommend mainly based on the distance from the user, as the other ‘low’ price is far enough as to be less relevant than distance.

The second customer has a preference for Fast Food, Cares about Rating (then Price), no time preference

```
        SELECT name, average, price
        FROM searching
        WHERE days LIKE '%' || 'Sat' || '%' AND endTime >= '22:00' AND startTime <= '22:00' AND
"Rcuisine"='Fast_Food'
        ORDER BY average DESC, price;
```



	name	average	price
1	Tortas Locas Hipocampo	1.1666666666666667	medium
2	Subway	0.88888888888888888889	low
3	La Fontana Pizza Restaurante and Cafe	0.875	high
4	pizza clasica	0.77777777777777777778	medium

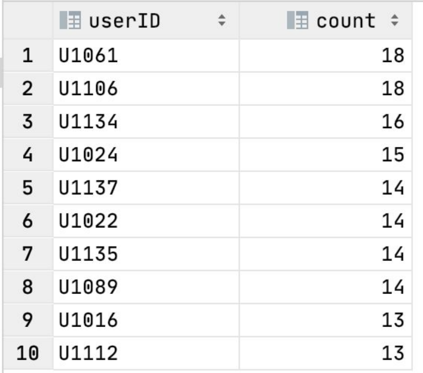
Looking at the results of this query, the recommended three restaurants would be Tortas Locas Hipocampo, Subway, and La Fontana Pizza. As this customer has a preference for rating over price, we decide to recommend one restaurant of each price range.

d) First I found the top 10 people who give the most reviews:

Query:

```
SELECT "userID", count(*)
FROM rating_final
GROUP BY "userID"
ORDER BY 2 DESC
LIMIT 10
```

Top reviewers:



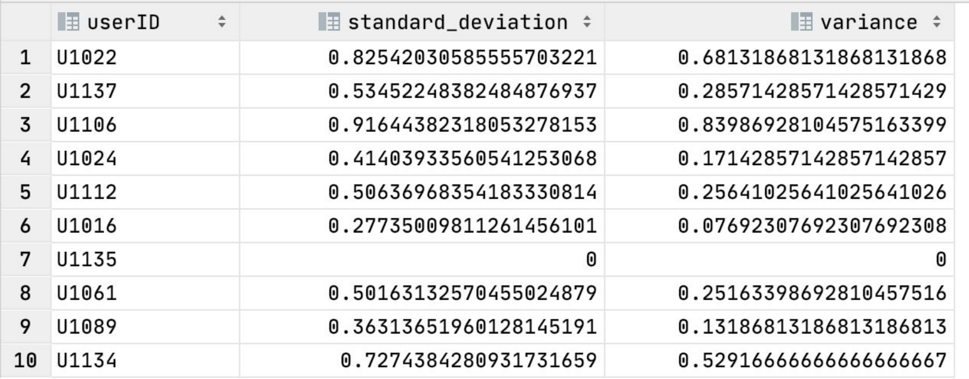
	userID	count
1	U1061	18
2	U1106	18
3	U1134	16
4	U1024	15
5	U1137	14
6	U1022	14
7	U1135	14
8	U1089	14
9	U1016	13
10	U1112	13

Then I found the variance and standard deviation of the reviews given by the top reviewers above. These statistics give us an idea of the diversity of reviews given by the reviews. Greater variance and standard deviation means the reviewer gave very different reviews, whereas low standard deviation and variance signals the reviewer was very consistent in the level of review they gave.

Query:

```
SELECT *
FROM (SELECT "userID", stddev(rating) as standard_deviation,
            variance(rating)
    FROM (SELECT "userID", rating
        FROM rating_final
        WHERE "userID" IN (SELECT "userID"
                        FROM (SELECT "userID", count(*)
                            FROM rating_final
                            GROUP BY "userID"
                            ORDER BY 2 DESC
                            LIMIT 10) AS K)) AS R
GROUP BY "userID") AS Q
```

Output:



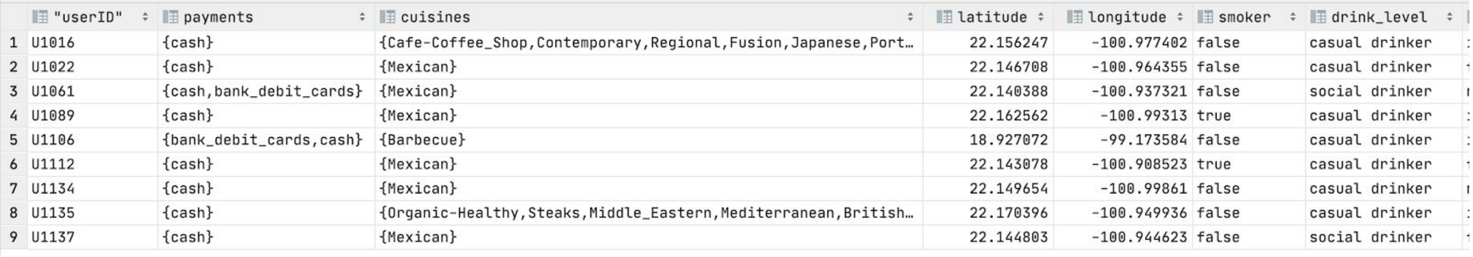
	userID	standard_deviation	variance
1	U1022	0.82542030585555703221	0.68131868131868131868
2	U1137	0.53452248382484876937	0.28571428571428571429
3	U1106	0.91644382318053278153	0.83986928104575163399
4	U1024	0.41403933560541253068	0.17142857142857142857
5	U1112	0.50636968354183330814	0.25641025641025641026
6	U1016	0.27735009811261456101	0.07692307692307692308
7	U1135	0	0
8	U1061	0.50163132570455024879	0.25163398692810457516
9	U1089	0.36313651960128145191	0.13186813186813186813
10	U1134	0.7274384280931731659	0.52916666666666666667

Lastly, finding all the info on the reviews is quite easy as we can just find the reviewers in the user table. However, one thing to note is that the way we have set up our user table was using inner joins. By doing this we eliminated cases where a user is not present in one of the original user tables. For instance if a user is present in userprofile, but they are not present in userpayment, then we have made the choice that they will not be present in the users table. This only eliminated 5 tuples so we felt it was a good design situation which prevented cases of nulls, however in this case one of the top reviewers in the rating_final table does not appear in our users table so we only have info on the top 9.

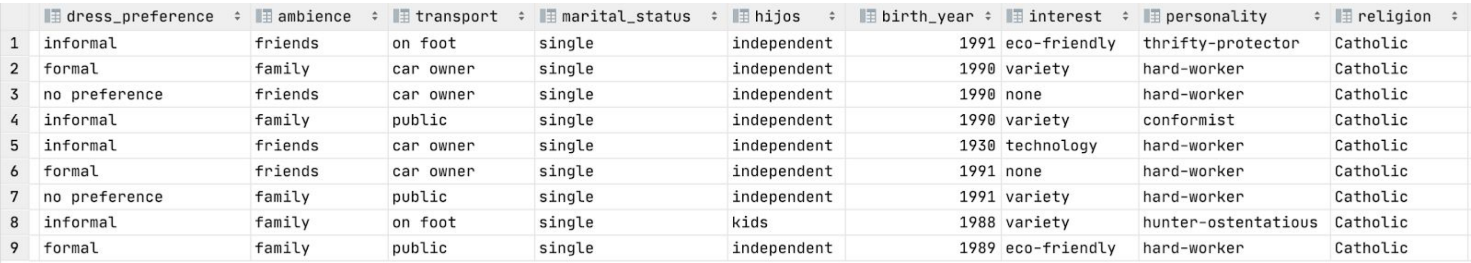
Query:

```
SELECT *
FROM users
WHERE "userID" IN (SELECT "userID"
        FROM (SELECT "userID", count(*)
            FROM rating_final
            GROUP BY "userID"
            ORDER BY 2 DESC
            LIMIT 10) AS K)
```

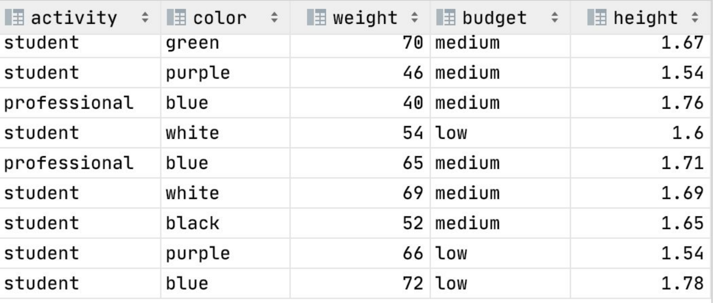
Output for info on top reviewers:



	"userID"	payments	cuisines	latitude	longitude	smoker	drink_level
1	U1016	{cash}	{Cafe-Coffee_Shop,Contemporary,Regional,Fusion,Japanese,Port…	22.156247	-100.977402	false	casual drinker
2	U1022	{cash}	{Mexican}	22.146708	-100.964355	false	casual drinker
3	U1061	{cash,bank_debit_cards}	{Mexican}	22.140388	-100.937321	false	social drinker
4	U1089	{cash}	{Mexican}	22.162562	-100.99313	true	casual drinker
5	U1106	{bank_debit_cards,cash}	{Barbecue}	18.927072	-99.173584	false	casual drinker
6	U1112	{cash}	{Mexican}	22.143078	-100.908523	true	casual drinker
7	U1134	{cash}	{Mexican}	22.149654	-100.99861	false	casual drinker
8	U1135	{cash}	{Organic-Healthy,Steaks,Middle_Eastern,Mediterranean,British…	22.170396	-100.949936	false	casual drinker
9	U1137	{cash}	{Mexican}	22.144803	-100.944623	false	social drinker



	dress_preference	ambience	transport	marital_status	hijos	birth_year	interest	personality	religion
1	informal	friends	on foot	single	independent	1991	eco-friendly	thrifty-protector	Catholic
2	formal	family	car owner	single	independent	1990	variety	hard-worker	Catholic
3	no preference	friends	car owner	single	independent	1990	none	hard-worker	Catholic
4	informal	family	public	single	independent	1990	variety	conformist	Catholic
5	informal	friends	car owner	single	independent	1930	technology	hard-worker	Catholic
6	formal	friends	car owner	single	independent	1991	none	hard-worker	Catholic
7	no preference	family	public	single	independent	1991	variety	hard-worker	Catholic
8	informal	family	on foot	single	kids	1988	variety	hunter-ostentatious	Catholic
9	formal	family	public	single	independent	1989	eco-friendly	hard-worker	Catholic



activity	color	weight	budget	height
student	green	70	medium	1.67
student	purple	46	medium	1.54
professional	blue	40	medium	1.76
student	white	54	low	1.6
professional	blue	65	medium	1.71
student	white	69	medium	1.69
student	black	52	medium	1.65
student	purple	66	low	1.54
student	blue	72	low	1.78

e) Logic: Our logic as investors is straightforward. We are assuming that high reviews reflect the cuisine, price level, city, and type of restaurant that will do well. We will examine different rating statistics between those four groups in order to find the best options for a new restaurant.

Here is the view that will add cuisine, price level, and city to the final rating table. I used inner joins so as to prevent any nulls in the columns.

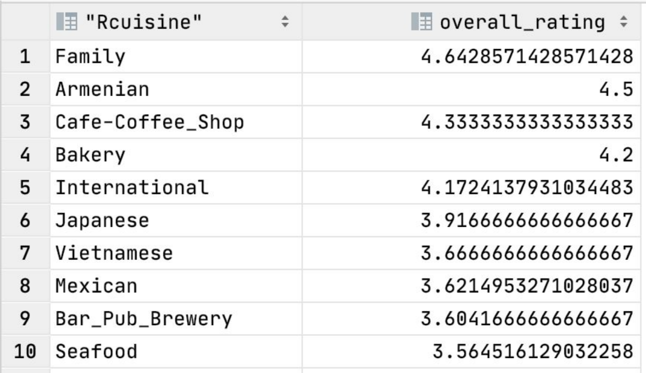
```
CREATE VIEW reviews AS
SELECT "userID", rating_final."placeID", rating, food_rating,
    service_rating, "Rcuisine", city, price, "Rambience"
FROM rating_final
INNER JOIN (SELECT DISTINCT "placeID", "Rcuisine"
     FROM chefmoz
     ORDER BY "placeID") K
ON rating_final."placeID" = K."placeID"
INNER JOIN geoplaces2 g on rating_final."placeID" = g."placeID"
ORDER BY "userID"
```

i) As an investor, first we are interested in knowing which type of cuisine is popular amongst reviewers. By figuring out the highest rated cuisines we can have a good idea of what type of cuisine we will want our restaurant to serve. Our logic is that the highest rated cuisine tends to be the cuisine that people want to eat more often. We did this by adding cuisine to the rating final table then averaging the different ratings amongst the different cuisines. Here are our query and results:

Query:

```
SELECT "Rcuisine", (avg_food + avg_rating + service_rating) AS overall_rating
FROM (SELECT "Rcuisine", avg(rating) avg_rating, avg(food_rating)
     AS avg_food, avg(service_rating) AS service_rating
  FROM reviews
  GROUP BY "Rcuisine") K
ORDER BY overall_rating DESC
```

Results:



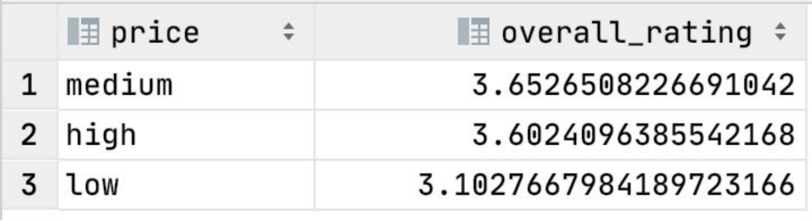
	"Rcuisine"	overall_rating
1	Family	4.6428571428571428
2	Armenian	4.5
3	Cafe-Coffee_Shop	4.3333333333333333
4	Bakery	4.2
5	International	4.1724137931034483
6	Japanese	3.9166666666666667
7	Vietnamese	3.6666666666666667
8	Mexican	3.6214953271028037
9	Bar_Pub_Brewery	3.6041666666666667
10	Seafood	3.564516129032258

ii) Next we were interested in knowing which price level are reviewers favoring the most. Our intuition tells us that the cheaper the food the better. However, by looking at the overall rating, defined the same as above, we can verify this assumption. Here is our query and results:

Query:

```
SELECT price, (avg_rating + service_rating + avg_food) AS overall_rating
FROM (SELECT "price", avg(rating) avg_rating, avg(food_rating)
      AS avg_food, avg(service_rating) AS service_rating
  FROM reviews
  GROUP BY price) AS K
ORDER BY overall_rating DESC
```

Results:



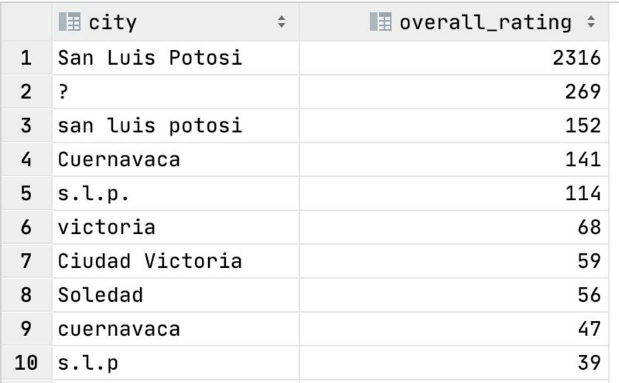
	price	overall_rating
1	medium	3.6526508226691042
2	high	3.6024096385542168
3	low	3.1027667984189723166

iii) Now that we have found the cuisines that are well liked and the appropriate price level, we want to know which city offers the best scene for restaurants. Our logic is that the city with the highest overall rating will be the best for new restaurants. However, we are interested in knowing which city has the best rated restaurants, but not cities where there happens to be one restaurant with good ratings, but no other restaurants. So instead of looking at the average ratings then adding them up, we looked at the sum of ratings for each city. This way we know which cities have many good reviews. However, we don't want to find cities with just way more reviews than other cities that they appear to have good reviews, even though there might be many bad ones. To prevent this we also look at the average rating within each city. So in the end we are most interested in cities which offer a good balance between not too few reviews, along with a high average rating. Here are our results:

Query 1:

```
SELECT city, (avg_food + avg_rating + service_rating) AS overall_rating
FROM (SELECT city, sum(rating) avg_rating, sum(food_rating)
     AS avg_food, sum(service_rating) AS service_rating, count(*) AS total
  FROM reviews
  GROUP BY city) K
ORDER BY overall_rating DESC
```

Results 1:

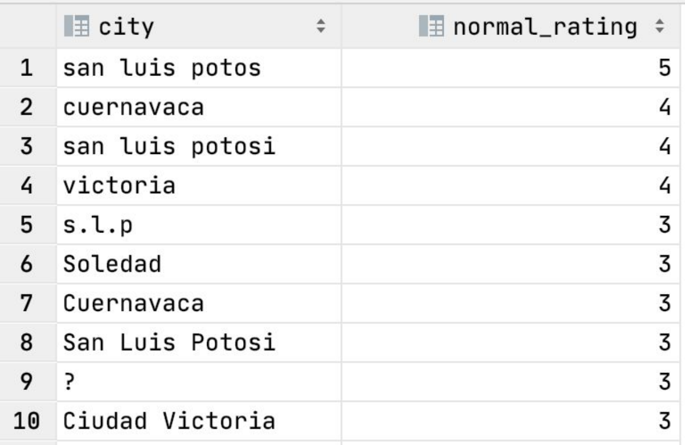


	city	overall_rating
1	San Luis Potosi	2316
2	?	269
3	san luis potosi	152
4	Cuernavaca	141
5	s.l.p.	114
6	victoria	68
7	Ciudad Victoria	59
8	Soledad	56
9	cuernavaca	47
10	s.l.p	39

Query 2:

```
SELECT city, ((avg_food + avg_rating + service_rating) / total) AS normal_rating
FROM (SELECT city, sum(rating) avg_rating, sum(food_rating)
   AS avg_food, sum(service_rating) AS service_rating, count(*) AS total
 FROM reviews
 GROUP BY city) K
ORDER BY normal_rating DESC
```

Results 2:

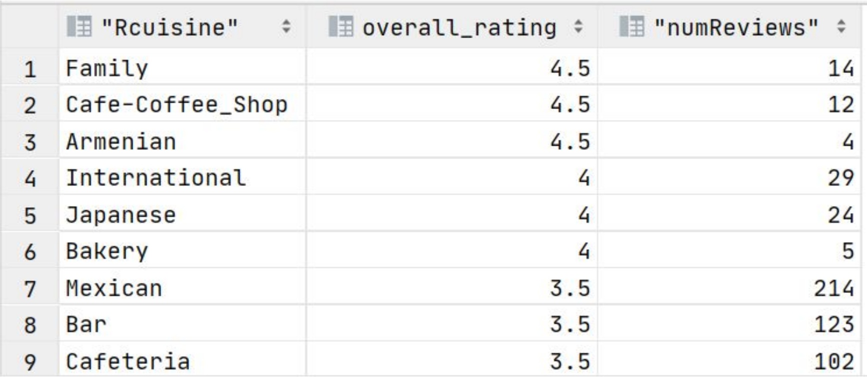


	city	normal_rating
1	san luis potos	5
2	cuernavaca	4
3	san luis potosi	4
4	victoria	4
5	s.l.p	3
6	Soledad	3
7	Cuernavaca	3
8	San Luis Potosi	3
9	?	3
10	Ciudad Victoria	3

iv) As we get closer to our final decision, we revisit our analysis of the popularity of different cuisines, this time through the lens of number of reviews. Our reasoning is that in order for a user to have left a review of a given restaurant, they must have visited the restaurant and felt strongly enough about the meal to post it online- This gives us insight into how many people frequent the different options of cuisine, and allows us to decide on the cuisine of our restaurant. We chose to round the average ratings of the restaurant in order to be able to ORDER BY two attributes.

Query 1:

```
SELECT "Rcuisine", round((avg_food + avg_rating + service_rating)*2, 0) / 2 AS overall_rating, review as
"numReviews"
FROM (SELECT "Rcuisine", avg(rating) avg_rating, avg(food_rating)
   AS avg_food, avg(service_rating) AS service_rating, count("userID") AS review
 FROM reviews
 GROUP BY "Rcuisine") K
ORDER BY "overall_rating" DESC, "numReviews" DESC;
```

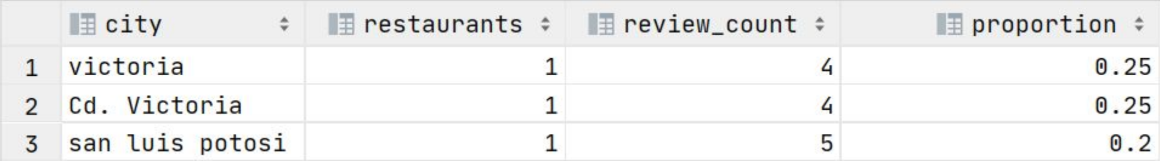


	"Rcuisine"	overall_rating	"numReviews"
1	Family	4.5	14
2	Cafe-Coffee_Shop	4.5	12
3	Armenian	4.5	4
4	International	4	29
5	Japanese	4	24
6	Bakery	4	5
7	Mexican	3.5	214
8	Bar	3.5	123
9	Cafeteria	3.5	102

Looking at this table, we have a few viable options to choose from. Family-Owned, Cafes, and Armenian food all have a rounded rating of 4.5, but only the first two have more than a few reviews (A few being subjective, but choosing a minimum number of reviews to be 10 allows us to narrow down our choices). In addition, International and Japanese food have high ratings and number of reviews.

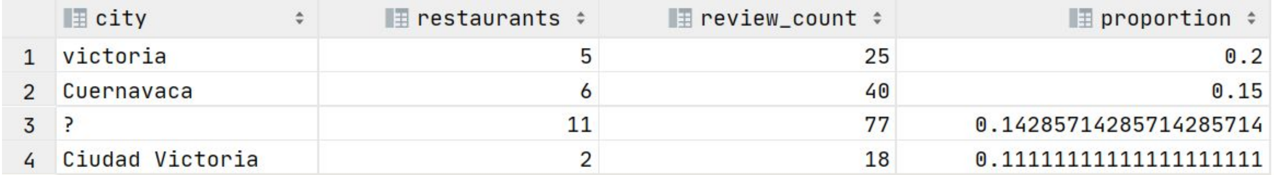
v) Lastly, we took a look at how many preexisting restaurants there were per city, and compared it to the number of reviews of restaurants in that city. By sorting by this proportion we are able to see what cities have a large number of reviews and few restaurants, possibly indicating the existence of an open spot for us in the market of that city. We took two looks at the data, one simply sorted, and one filtering out cities with a review count of under 10, in order to avoid skewing the proportion with cities with a single restaurant.

```
SELECT R.city, count(DISTINCT R."placeID") AS "restaurants",
    count(R."placeID") AS "review_count",
    CAST(count(DISTINCT R."placeID") AS DECIMAL) / CAST(count(R."placeID") AS DECIMAL) AS
"proportion"
FROM reviews R
GROUP BY R.city
ORDER BY "proportion" DESC;
```



	city	restaurants	review_count	proportion
1	victoria	1	4	0.25
2	Cd. Victoria	1	4	0.25
3	san luis potosi	1	5	0.2

```
SELECT *
FROM (SELECT R.city, count(DISTINCT R."placeID") AS "restaurants",
    count(R."placeID") AS "review_count",
    CAST(count(DISTINCT R."placeID") AS DECIMAL) / CAST(count(R."placeID") AS DECIMAL) AS
"proportion"
FROM reviews R
GROUP BY R.city) AS K
WHERE "review_count" >= 10
ORDER BY "proportion" DESC;
```



	city	restaurants	review_count	proportion
1	victoria	5	25	0.2
2	Cuernavaca	6	40	0.15
3	?	11	77	0.14285714285714285714
4	Ciudad Victoria	2	18	0.11111111111111111111

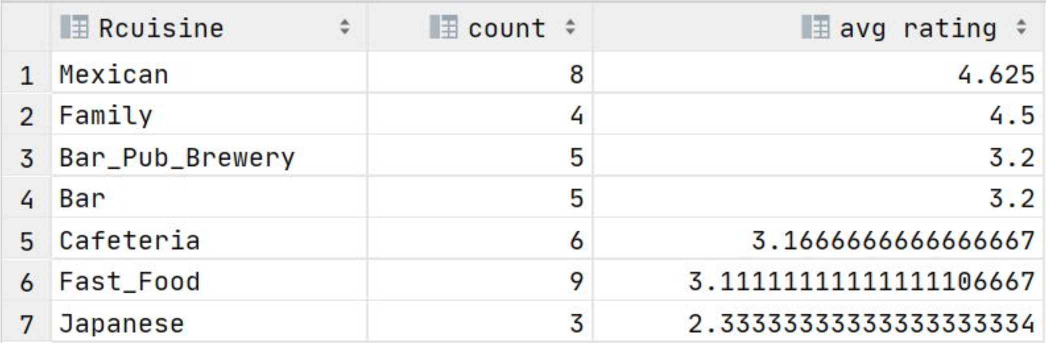
While the first query is not especially useful due to our hypothesized outliers, the second gives us a few options for highly-reviewed, under-represented cities. Victoria and Cuernavaca stand out as being popular despite the lack of restaurants. The ? city simply represents a restaurant that doesn't list its city of origin, or is otherwise mobile. Therefore, we can ignore it in favor of the other cities noted here.

Both of these cities, Victoria and Cuernavaca have a normalized rating of 4, though the latter has a larger total rating due to having more reviews. However, the way our dataset is organized hints at there being multiple possible ‘city’ values for the same city, spelled different ways, i.e. victoria vs ciudad victoria. Given this, and the total values being similar, we select Cuernavaca as our choice of City in order to simplify future analysis we would do on the area.

vi) To decide which type of cuisine to use, we made one final query of the count and average rating of the different types of cuisine in Cuernavaca.

Query 1:

```
SELECT "Rcuisine", count("placeID"), AVG("rating") + AVG("food_rating") + AVG("service_rating") AS
"avg rating"
FROM reviews
WHERE city='Cuernavaca'
GROUP BY "Rcuisine"
ORDER BY 3 DESC;
```



	Rcuisine	count	avg rating
1	Mexican	8	4.625
2	Family	4	4.5
3	Bar_Pub_Brewery	5	3.2
4	Bar	5	3.2
5	Cafeteria	6	3.166666666666666667
6	Fast_Food	9	3.11111111111111106667
7	Japanese	3	2.33333333333333333334

This final table gives us that while both Family and Japanese Style restaurants are few in number, the former has a high average rating of 4.5, while the latter has a rating of only 2.33. With these two choices in mind, we have a dilemma. If we decide on a Family restaurant, the competition is likely fierce between these other highly-rated restaurants. However, deciding on a Japanese restaurant is also potentially unwise, as the low average might be an indication of the local area not being fond of that cuisine. With those two weights in mind, we decided that it’s much better to be diving into a competitive market than an unwanted market, particularly because of the prestigiousness by association to the highly-rated restaurants.

With these decisions in mind, our final decision on what to invest in is a Low-Priced (ii), Family (i, iii, vi) Restaurant based in Cuernavaca (iv, v).