Implementation of a Basic Restaurant Recommendation System

Md Azim Ullah
mullah@memphis.edu

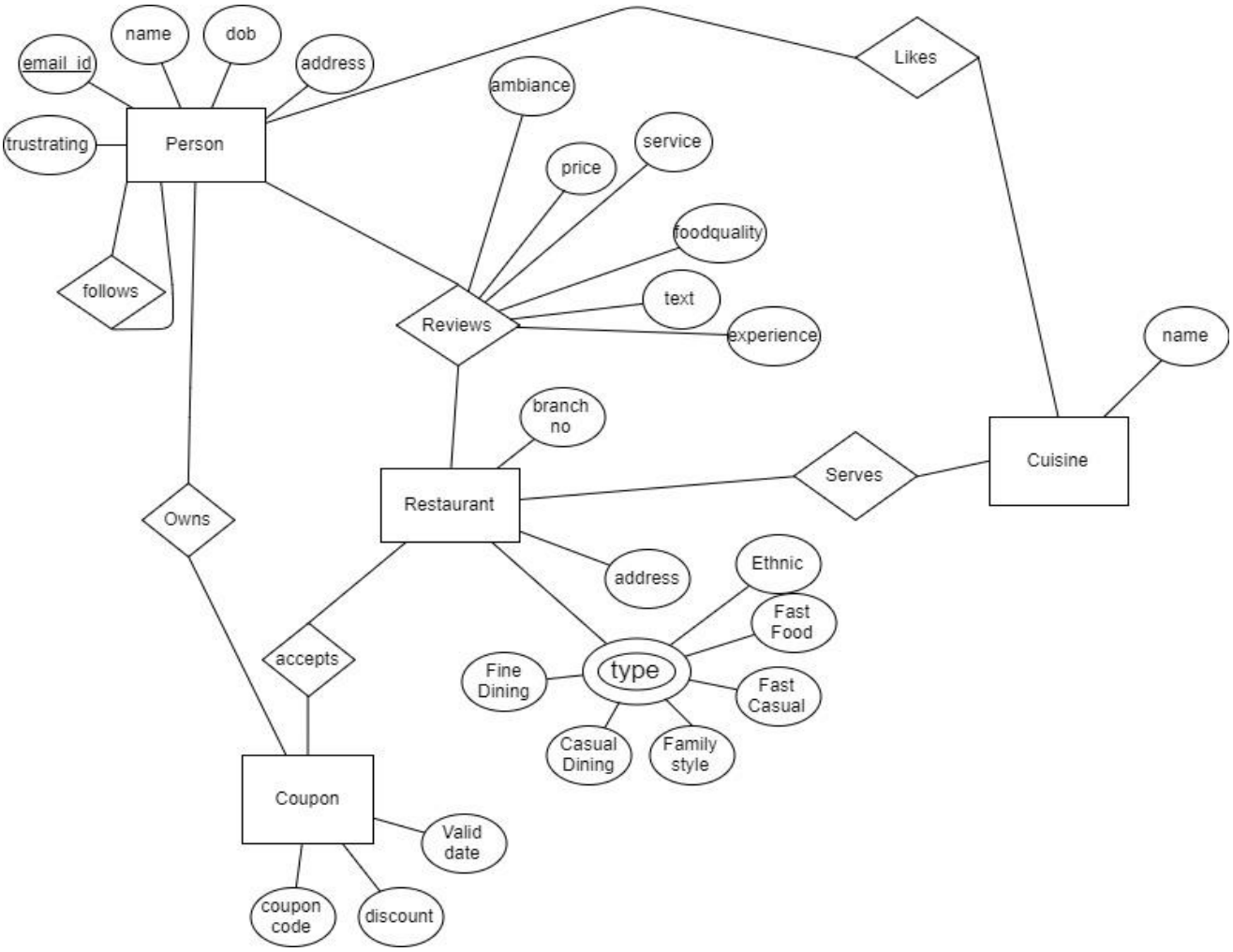
Introduction:

This is a paper detailing the implementation details of a basic restaurant recommendation system. The goal of this implementation was to get a working knowledge on relational database modelling. A database was built in PostgreSQL based on the constraints given. The front end was developed in python to connect to this database using the most widely used PostgreSQL adapter “psycopg” and get queried results. To build the simplest html based front end Flask web framework was used.

Architecture:

The given constraints were first mapped to a entity relationship(ER) model.

The ER model I found appropriate to freeze is



Several assumptions were made in this process such as

- A restaurant can be in multiple addresses which was assumed to be resulting from different branches of the same restaurant.
- Although email_id is unique to any person for convenience of integer matching email_id was not taken as primary key.
- Several M:N relationships were assumed between entities such as person follows person,restaurant has cuisines,person owns coupons etc some of which are evident from the given constraints but rest of them were made to account for generalized design in a realistic scenario.
- Cuisine is assumed to be a separate entity with persons and restaurants having 1:N and 1:M relationship to it.

This ER model was then mapped to a relational model as given below

Person

id	email_id	name	dob	address	trust_rating

Follow

follower_id	followed_id

Review

ambiance_score	service_score	price_score	experience_score	foodquality_score	text	person_id	restaurant_id

Restaurant

id	name	address	branch_no	type

Has_cuisines

restaurant_id	cuisine_id

Cuisine

id	name

Likes_cuisines

cuisine_id	person_id

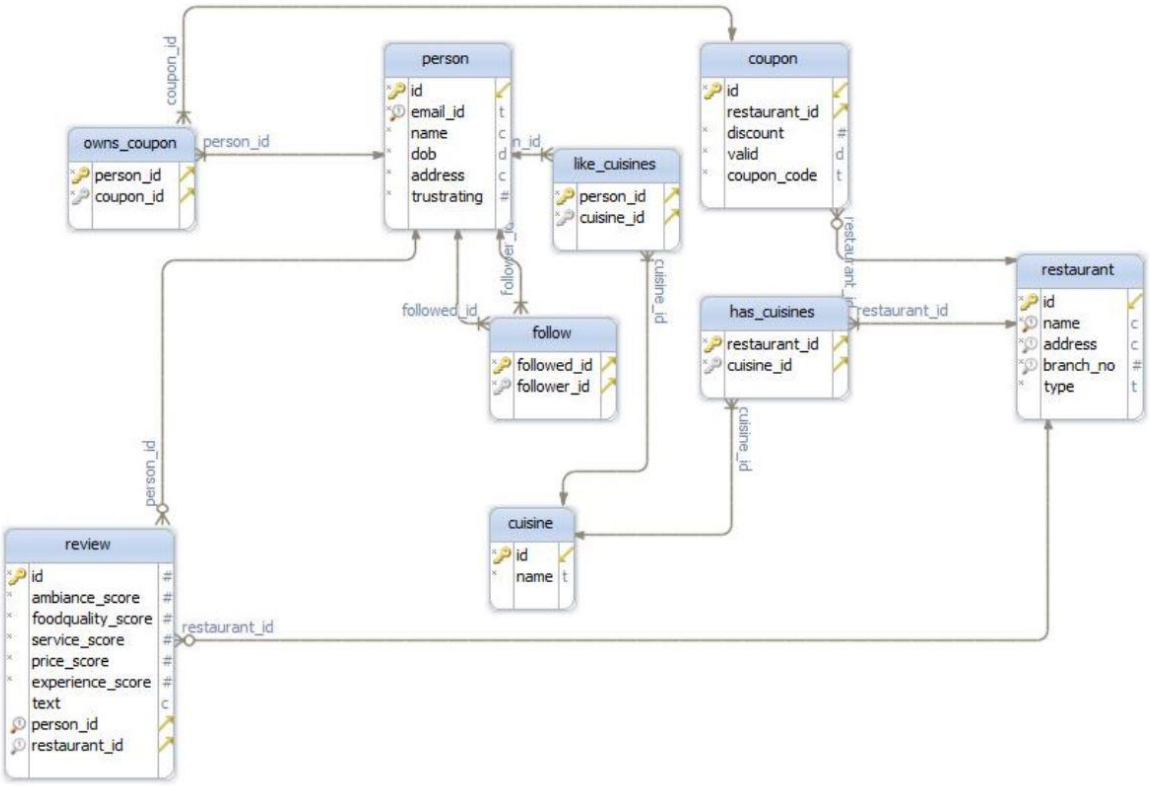
Coupon

id	restaurant_id	discount	valid	coupon_code

Owns_coupons

coupon_id	person_id

A more informative picture of the relational mapping is



Normalization:

As evident from the design surrogate keys were added for smooth implementation in different tables. The addition of these surrogate keys although violates 3NF in tables such as Person this implementation decision was taken to avoid string matching while joining the tables.

In Restaurant table “coupon_code” being an attribute functionally dependent on the foreign key “restaurant_id” violates 3NF. But the motivation was that coupon code is more specific to the coupon itself and is generally a character array that can be used to redeem the coupon. My design accounts for the fact that it is indeed the restaurant which circulates these coupons which has the coupon code embedded in them.

All the M:N relationships are devoid of surrogate keys since it does not add any specific advantage to the implementation.

SQL queries:

The four sql queries for the database are:

1. Find the name of the cuisines that a specific user = “A” likes and which restaurants are the available:

```
select cuisine.name, restaurant.name, restaurant.address from like_cuisines inner join
person on person.id=like_cuisines.person_id inner join cuisine on
cuisine.id=like_cuisines.cuisine_id inner join has_cuisines on has_cuisines.cuisine_id =
cuisine.id inner join restaurant on has_cuisines.restaurant_id = restaurant.id where
person.name='A';
```

2. Find all the valid coupons that user = “A” has and which restaurants are they specific to:

```
select restaurant.name,restaurant.address,coupon.discount::int,coupon.valid from person
inner join owns_coupon on owns_coupon.person_id = person.id inner join coupon on
coupon.id = owns_coupon.coupon_id inner join restaurant on
coupon.restaurant_id=restaurant.id where person.name= 'A' and coupon.valid > now()
order by coupon.discount desc;
```

3. Find the best restaurants as determined by the average of all kinds of ratings:

```
select restaurant.name,restaurant.address,(avg(review.ambiance_score)+
avg(review.price_score)+avg(review.foodquality_score)+avg(review.service_score)+
avg(review.experience_score))/5 as avg_score from restaurant inner join review on
review.restaurant_id = restaurant.id group by restaurant.id
order by avg_score desc;
```

4. Find the restaurant which offers the best value for money as determined by the average price rating given by users:

```
select restaurant.name,restaurant.address,avg(review.price_score)::int as price_rating
from restaurant inner join review on review.restaurant_id = restaurant.id group by
restaurant.id order by price_rating desc limit 1;
```

Limitations and Future Work:

I think the limitation of my work resides in the design of Coupon table. Users have coupons which can be used in restaurants. The database does not account for the fact that an attribute can be added which shows if it is already an used coupon or not. The use of surrogate keys was intelligently done but the effect of 3NF violation could be studied further.

In future I would plan to use the design to a full web based implementation of a restaurant system. A working restaurant website would be the pinnacle of this work which I hope I can achieve. Since the

Conclusion:

The project was meant to be a first-hand experience with a relational database. My decision to build a PostgreSQL database ensured I learnt many a relevant tool implementing the project. I believe this has been a worthy introduction to the practical aspects of relational database design.