CREATION OF AN ER MODEL, DESIGNING & CREATING THE SCHEMA

CS-322 Introduction to Database system

Semester project
Deliverable 1
Group 45

Instructors
Angelos Christos Anadiotis
Professor Christoph Koch

Assistant
Anna Herlihy

Students
Mark David Paul
Atassi Rami
Ribeiro de Carvalho Paulo

April 10, 2023

Contents

List of Figures

1 Introduction

This report details our approach to organizing data from the popular platform Yelp! We begin by describing the development of an Entity Relationship (ER) model based on real-world data sets, followed by the implementation of its relational schema. Throughout this process, we encountered a variety of data quality issues that required cleaning and transformation, which we discuss in the latter part of the report.

As with any data modeling, there were certain assumptions that guided our decisions. Some of these assumptions were based on hypothetical scenarios, while others were informed by an analysis conducted using a Python script. We provide detailed explanations of these assumptions and their implications throughout the report. In addition, our goal is to create a data structure that is both sufficiently optimized and user-friendly.

2 Entity Relationship Model

To start, we present our final Entity Relationship (ER) model in an informal, yet more accessible format. For practical and readability purposes, we opted to group together the attributes of each entity. For a more formal and detailed version of the ER model, please refer to the annex.

During our exploration of the data sets (with python script), we observed that the "Business" entity had an attribute known as "attributes" that grouped together various sub-attributes. In response, we decided to create separate entities for each of the sub-attributes, namely "GoodForMeal", "BusinessParking", "NoiseLevel", "Ambience", "Music", and "DietaryRestrictions". Additionally, we created two weak entities, "Localisation" and "Hours". The former was created to group together all geographic attributes, while the latter was based on the fact that the data in the "Hours" attribute was a collection of sub-attributes representing each day of the week. Our decision to create these entities was motivated by our desire to keep thing easy and clear and to ensure that every entity contained only atomic data, thereby promoting a more efficient and streamlined data organization.

Furthermore, these design choices optimize the storage and speed of future queries. For example, creating entities for the sub-attributes in "Business" significantly reduces the amount of data stored. Explanations for this will be provided in the report.

2.1 Simplified ER Model

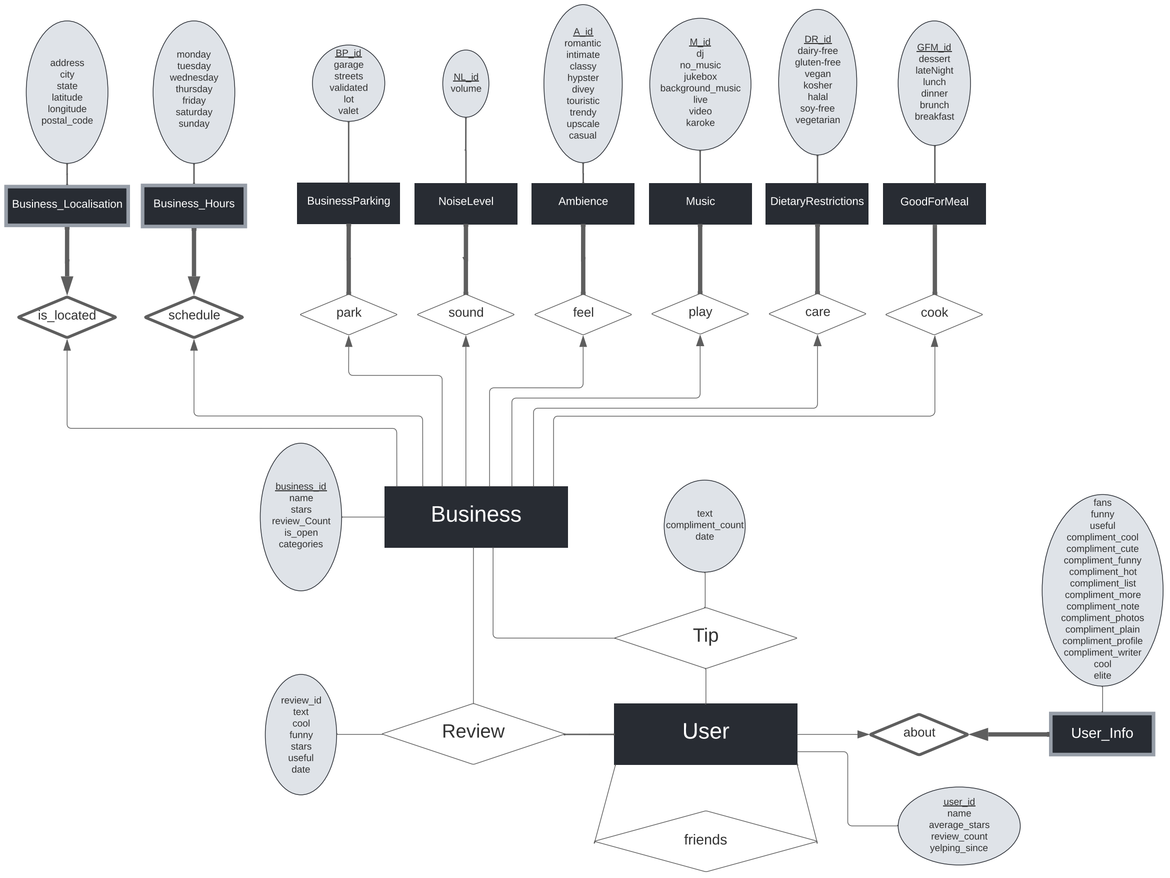


Figure 1: ER model simplified

2.2 Constraints choices

Constraints were established for each relation in our ER model, and we provide detailed explanations of these constraints in the subsequent paragraphs. Note that certain relations share the same constraints, and in these cases, we provide a single explanation for the sake of brevity and clarity.

2.2.1 Business/Business_Localisation Relation

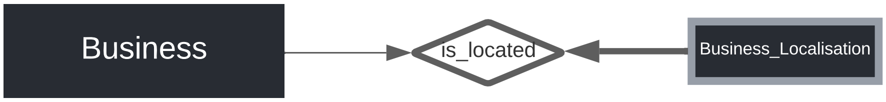


Figure 2: Business/Business_Localisation Relation

By running a simple Python script, we observed that not all businesses have an address recorded. Some are labeled as "Nan," but none have more than one address. This is not surprising, as a single business cannot be in two places at once. While a business can have multiple franchises with the same name, each will have a distinct business ID.

As a result, the data from "Business" is linked with at most one record in "Business_Localisation," as represented by an arrow. It is worth noting that every tuple in "Business_Localisation" has exactly one business to reference, as denoted by a bold arrow.

In this case, a weak entity is particularly useful, as it represents a parent-child relationship. If a parent entity (business) is deleted, its associated address no longer serves any purpose in our database and can be safely removed as well. It should be noted that the relationship between "Business" and "Business_Hour" follows the same rationale.

2.2.2 Business/Ambience Relation & co.

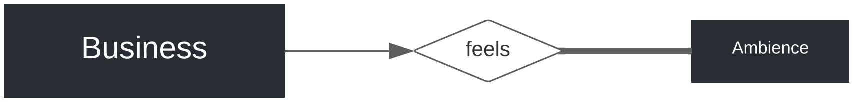


Figure 3: Business/Ambience Relation

Upon examining the "Ambience" entity, it becomes clear that each attribute can only have one of two Boolean values. Therefore, for each sample, a maximum of $2^9 = 512$ possible combinations can exist, taking into account all attribute (e.g. 9 in "Ambience" entity) variations. Each business can be linked to at most one record in the "Ambience" entity, represented by an arrow. On the other hand, each sample in "Ambience" can be linked to multiple businesses. If no business is linked to a sample, it is considered unnecessary and will be removed. Hence, each sample in "Ambience" needs to be linked to at least one business, represented by a bold line.

The participation constraint mentioned above requires a more complex and expensive operation to be verified, which we have not fully studied during the course. Therefore, it may not be optimal to do so. In this case, storing all possible combinations may be a good choice. It should be noted that the relationship between "Business" and all other sub-attributes ("GoodForMeal", "Music", and so on) follows the same rationale.

2.2.3 User/Review/Business Relation

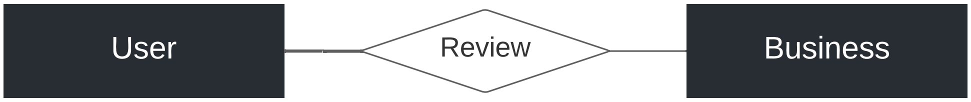


Figure 4: User/Review/Business Relation

During the design process of our ER model, we had an idea to use the "Review" dataset to create a relation between the "User" and "Business" entities, rather than creating a separate entity. Under our assumptions, no constraints are needed. It is possible for every user to write multiple reviews, but they can also choose not to write any reviews. Similarly, a business can have multiple reviews, but can also have none.

2.2.4 User/Tip/Business Relation

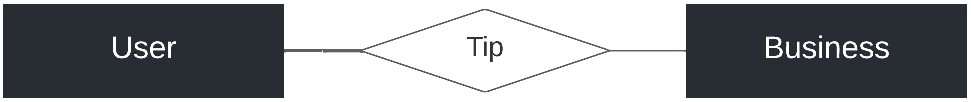


Figure 5: User/Tip/Business Relation

We have taken a similar approach for the "Tip" relation in our ER model. Given our assumptions, no constraints are needed for this relation. Every user can write multiple tips, but they can also choose not to write any. Similarly, a business can have multiple tips, but it can also have none.

2.2.5 User/User Relation

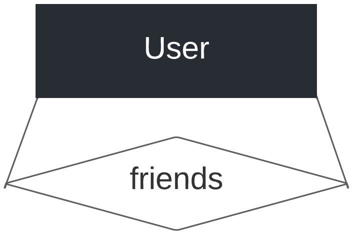


Figure 6: User/User Relation

The "friends" relation links multiple users, which means that one user can be linked to multiple others, but also to none at all. The same applies to the other direction. It is not necessary for every user to have friends to use the Yelp! platform, so no constraints are needed.

Although our Python script revealed that every user in the dataset has at least one friend, but we make the assumption that no participation constraints apply in our model. We believe it is impractical to require users to have friends, and our assumption offers more flexibility in designing the database schema.

2.2.6 User/User_Info Relation

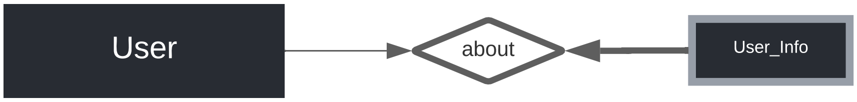


Figure 7: User/User_Info Relation

To simplify the "User" entity and make it more efficient, a weak entity named "User_info" was created to store detailed information about each user. This way, only the necessary information is stored in the main "User" entity. Each sample in the "User_info" entity is linked to exactly one user, represented by a bold arrow, while each user can be linked to at most one sample in the "User_info" entity, but may not be linked to any if no information is available. The weak entity also ensures that all information in "User_info" is deleted if its corresponding user is deleted.

3 Relational Schema

This section presents the Relational Schema for each relation in a clear format. A table is also provided to facilitate the understanding of the entities and their relationships. The constraints discussed above have been incorporated into the schema to the fullest extent possible.

3.1 Business

BUSINESS(***business_ID***:string, ***name***:string, ***stars***:float, ***review_count***:integer, ***is_open***:integer, ***categories***:string)

Foreign key : *None*

```
CREATE TABLE Business(
    business_ID CHAR(22),
    name CHAR(100),
    stars NUMBER(2,1),
    review_count INTEGER,
    is_open NUMBER(1),
    categories CHAR(500)
   PRIMARY KEY(business_ID))
```

business_ID	name	stars	review_count	is_open	categories

3.2 Business_Localisation

BUSINESS_LOCALISATION(***business_ID***:string, ***adress***:string, ***city***:string, ***state***:string, ***latitude***:float, ***longitude***:float, ***postal_code***:string)

Foreign key : *business_ID*

```
CREATE TABLE Business_Localisation(
    business_ID CHAR(22),
    adress CHAR(100),
    city CHAR(30),
    state CHAR(2),
    latitude FLOAT,
    longitude FLOAT,
    postal_code CHAR(30),
   PRIMARY KEY(business_ID),
   FOREIGN KEY(business_ID) REFERENCES Business(business_ID)
       ON DELETE CASCADE)
```

business_ID	adress	city	state	latitude	longitude	postal_code

3.3 Business_Hours

BUSINESS_HOURS(***business_ID***:string, ***monday***:string, ***tuesday***:string, ***wednesday***:string, ***thursday***:string, ***friday***:string, ***saturday***:string, ***sunday***:string)

Foreign keys : *business_ID*

```
CREATE TABLE Business_Hours(
     business_ID CHAR(22),
     monday CHAR(20),
     tuesday CHAR(20),
     wednesday CHAR(20),
     thursday CHAR(20),
     friday CHAR(20),
     saturday CHAR(20),
     sunday CHAR(20),
     PRIMARY KEY(business_ID),
     FOREIGN KEY(business_ID) REFERENCES Business(business_ID)
          ON DELETE CASCADE)
```

business_ID	monday	tuesday	wednesday	thursday	friday	saturday	sunday

3.4 GoodForMeal

GOODFORMEAL(***GFM_ID***:string, ***dessert***:int, ***late_night***:int, ***lunch***:int, ***dinner***:int, ***brunch***:int, ***breakfast***:int)

Foreign key : *None*

```
CREATE TABLE GoodForMeal(
     GFM_ID CHAR(10),
     dessert NUMBER(1),
     late_night NUMBER(1),
     lunch NUMBER(1),
     dinner NUMBER(1),
     brunch NUMBER(1),
     breakfast NUMBER(1),
     PRIMARY KEY(GFM_ID))
```

GFM_ID	dessert	late_night	lunch	dinner	brunch	breakfast

3.5 Cook

COOK(***business_ID***:string, ***GFM_ID***:string)

Foreign key : *GFM_ID* & *business_ID*

```
CREATE TABLE Cook(
     business_ID CHAR(22),
     GFM_ID CHAR(10) NOT NULL,
     PRIMARY KEY(business_ID),
     FOREIGN KEY(GFM_ID) REFERENCES GoodForMeal(GFM_ID),
     FOREIGN KEY(business_ID) REFERENCES Business(business_ID)
          ON DELETE CASCADE))
```

business_ID	GFM_ID

3.6 BusinessParking

BUSINESSPARKING(***PB_ID***:string, ***garage***:integer, ***street***:integer, ***validated***:integer, ***lot***:integer, ***valet***:integer)

Foreign key : *None*

```
CREATE TABLE BusinessParking(
     PB_ID CHAR(10),
     garage NUMBER(1),
     street NUMBER(1),
     validated NUMBER(1),
     lot NUMBER(1),
     valet NUMBER(1),
     PRIMARY KEY(BP_ID))
```

PB_ID	garage	street	validated	lot	valet

3.7 Park

PARK(***business_ID***:string, ***PB_ID***:string)

Foreign key : *PB_ID* & *business_ID*

```
CREATE TABLE Park(
     business_ID CHAR(22),
     PB_ID CHAR(10) NOT NULL,
     PRIMARY KEY(business_ID),
     FOREIGN KEY(PB_ID) REFERENCES BusinessParking(PB_ID),
     FOREIGN KEY(business_ID) REFERENCES Business(business_ID)
          ON DELETE CASCADE)
```

business_ID	BP_ID

3.8 NoiseLevel

NOISELEVEL(***NL_ID***:string, ***level***:string)

Foreign key : *None*

```
CREATE TABLE NoiseLevel(
     NL_ID CHAR(10),
     level CHAR(30),
     PRIMARY KEY(NL_ID))
```

NL_ID	level

3.9 Sound

SOUND(***business_ID***:string, ***NL_ID***:string)

Foreign key : *NL_ID & business_ID*

```
CREATE TABLE Sound(
     business_ID CHAR(22),
     NL_ID CHAR(10) NOT NULL,
     PRIMARY KEY(business_ID),
     FOREIGN KEY(NL_ID) REFERENCES BusinessParking(NL_ID),
     FOREIGN KEY(business_ID) REFERENCES Business(business_ID)
          ON DELETE CASCADE)
```

business_ID	NL_ID

3.10 Ambience

AMBIENCE(***A_ID***:string, ***romantic***:integer, ***intimate***:integer, ***classy***:integer, ***hipster***:integer, ***divey***:integer, ***touristy***:integer, ***trendy***:integer, ***upscale***:integer, ***casual***:integer)

Foreign key : *None*

```
CREATE TABLE Ambience(
     A_ID CHAR(10),
     romantic NUMBER(1),
     intimate NUMBER(1),
     classy NUMBER(1),
     hipster NUMBER(1),
     divey NUMBER(1),
     touristy NUMBER(1),
     trendy NUMBER(1),
     upscale NUMBER(1),
     casual NUMBER(1),
     PRIMARY KEY(A_ID))
```

A_ID	romantic	intimate	classy	hipster	divey	touristy	trendy	upscale	casual

3.11 Feel

FEEL(***business_ID***:string, ***A_ID***:string)

Foreign key : *A_ID & business_ID*

```
CREATE TABLE Feel(
     business_ID CHAR(22),
     A_ID CHAR(10) NOT NULL,
     PRIMARY KEY(business_ID),
     FOREIGN KEY(A_ID) REFERENCES BusinessParking(A_ID),
     FOREIGN KEY(business_ID) REFERENCES Business(business_ID)
          ON DELETE CASCADE)
```

business_ID	A_ID

3.12 Music

MUSIC(***M_ID*:string**, ***dj***:integer, ***background_music***:integer, ***no_music***:integer, ***jukebox***:integer, ***live***:integer, ***video***:integer, ***karaoke***:integer)

Foreign key : *None*

```
CREATE TABLE Music(
    M_ID CHAR(10),
    dj NUMBER(1),
    background_music NUMBER(1),
    no_music NUMBER(1),
    jukebox NUMBER(1),
    live NUMBER(1),
    video NUMBER(1),
    karaoke NUMBER(1),
    PRIMARY KEY(M_ID))
```

M_ID	dj	background_music	no_music	jukebox	live	video	karaoke

3.13 Play

PLAY(***business_ID*:string**, ***M_ID***:string)

Foreign key : *M_ID* & *business_ID*

```
CREATE TABLE Play(
    business_ID CHAR(22),
    M_ID CHAR(10) NOT NULL,
    PRIMARY KEY(business_ID),
    FOREIGN KEY(M_ID) REFERENCES BusinessParking(M_ID),
    FOREIGN KEY(business_ID) REFERENCES Business(business_ID)
        ON DELETE CASCADE)
```

business_ID	M_ID

3.14 DietaryRestrictions

DIETARYRESTRICTIONS(***DR_ID*:string**, ***dairy-free***:integer, ***gluten-free***:integer, ***vegan***:integer, ***kosher***:integer, ***halal***:integer, ***soy-free***:integer, ***vegetarian***:integer)

Foreign key : *None*

```
CREATE TABLE DietaryRestrictions(
    DR_ID CHAR(10),
    dairy-free NUMBER(1),
    gluten-free NUMBER(1),
    vegan NUMBER(1),
    kosher NUMBER(1),
    halal NUMBER(1),
    soy-free NUMBER(1),
    vegetarian NUMBER(1),
    PRIMARY KEY(DR_ID))
```

DR_ID	dairy-free	gluten-free	vegan	kosher	halal	soy-free	vegetarian

3.15 Care

CARE(***business_ID***:string, ***DR_ID***:string)

Foreign key : *DR_ID* & *business_ID*

```
CREATE TABLE Care(
    business_ID CHAR(22),
    DR_ID CHAR(10) NOT NULL,
    PRIMARY KEY(business_ID),
    FOREIGN KEY(DR_ID) REFERENCES BusinessParking(DR_ID),
    FOREIGN KEY(business_ID) REFERENCES Business(business_ID)
        ON DELETE CASCADE)
```

business_ID	DR_ID

3.16 User

USER(***user_ID***:string, ***name***:string, ***average_stars***:float, ***review_count***:integer, ***yelping_since***:date)

Foreign key : *None*

```
CREATE TABLE User(
    user_ID CHAR(22),
    name CHAR(100),
    average_stars NUMBER(3,2),
    review_count INTEGER,
    yelping_since DATE,
    PRIMARY KEY(user_ID))
```

user_ID	name	average_stars	review_count	yelping_since

3.17 User_Info

USER_HOURS(***user_ID***:string, ***compliment_cool***:integer, ***compliment_cute***:integer, ***compliment_funny***:integer, ***compliment_hot***:integer, ***compliment_list***:integer, ***compliment_more***:integer, ***compliment_note***:integer, ***compliment_photos***:integer, ***compliment_plain***:integer, ***compliment_profile***:integer, ***compliment_writer***:integer, ***cool***:integer, ***fans***:integer, ***funny***:integer, ***useful***:integer, ***elite***:integer)

Foreign key : *user_ID*

```
CREATE TABLE User_Info(
    user_ID CHAR(22),
    compliment_cool INTEGER,
    compliment_cute INTEGER,
    compliment_funny INTEGER,
    compliment_hot INTEGER,
    compliment_list INTEGER,
    compliment_more INTEGER,
    compliment_note INTEGER,
    compliment_photos INTEGER,
    compliment_plain INTEGER,
    compliment_profile INTEGER,
    compliment_writer INTEGER,
    cool INTEGER,
    fans INTEGER,
    funny INTEGER,
    useful INTEGER,
    elite INTEGER,
    PRIMARY KEY(user_ID),
    FOREIGN KEY(user_ID) REFERENCES User(user_ID)
        ON DELETE CASCADE)
```

user_ID	compliment_cool	compliment_cute	compliment_funny	compliment_hot	compliment_list	compliment_more	compliment_note	compliment_photos	compliment_plain	compliment_profile	compliment_writer	cool	elite	fans	funny	useful

3.18 Friends

FRIENDS(***user1_ID***:string, ***user2_ID***:string)

Foreign keys : *user1_ID* & *user2_ID*

```
CREATE TABLE Friends(
    user1_ID CHAR(22),
    user2_ID CHAR(22),
    PRIMARY KEY(user1_ID, user2_ID),
    FOREIGN KEY(user1_ID) REFERENCES User(user_ID)
        ON DELETE CASCADE,
    FOREIGN KEY(user2_ID) REFERENCES User(user_ID)
        ON DELETE CASCADE))
```

user1_ID	user2_ID

3.19 Tip

TIP(***user_ID***:string, ***date***:date, ***business_ID***:string, ***text***:string, ***compliment_count***:integer)

Foreign keys : *user_ID* & *business_ID*

```
CREATE TABLE Tip(
     user_ID CHAR(22),
     business_ID CHAR(22) NOT NULL,
     date DATE,
     text CHAR(10000),
     compliment_count INTEGER,
     PRIMARY KEY(user_ID, date),
     FOREIGN KEY(user_ID) REFERENCES User(user_ID),
     FOREIGN KEY(business_ID) REFERENCES Business(business_ID))
```

user_ID	business_ID	date	text	compliment_count

3.20 Review

REVIEW(***review_ID***:string ***user_ID***:string, ***business_ID***:string, ***date***:date, ***text***:string, ***cool***:integer, ***funny***:float, ***stars***:float, ***useful***:float)

Foreign keys : *user_ID* & *business_ID*

```
CREATE TABLE Review(
     review_ID CHAR(22),
     user_ID CHAR(22) NOT NULL,
     business_ID CHAR(22) NOT NULL,
     date DATE,
     text CHAR(10000),
     cool INTEGER,
     funny FLOAT,
     stars FLOAT,
     useful FLOAT,
     PRIMARY KEY(review_ID),
     FOREIGN KEY(user_ID) REFERENCES User(user_ID),
     FOREIGN KEY(business_ID) REFERENCES Business(business_ID))
```

review_ID	user_ID	business_ID	date	text	cool	funny	stars	useful

4 Data Cleaning and Transformation Discussion

During our data exploration, we encountered several issues, including inconsistency in data types. Specifically, some attributes contained multiple types of objects (e.g., float and string), which required us to standardize the format of all objects within the same attribute to avoid domain conflicts.

In some cases, a sample may have one or more attributes without any recorded values. These missing values are often represented as "None" or "NaN", but sometimes an empty list (e.g. "Attributes" column in "Business" dataset) or a space character may be used. This inconsistency in representing missing data can be problematic. Therefore, it would be better to have a single value that consistently represents the absence of data for an attribute.

We also observed inconsistencies in the structure of data within the same attribute. For instance, in the "Attributes" attribute of the "Business" data set, we found variations in the way data was written, which posed a challenge in ensuring structural consistency across the data set. The data may begin with either "'BusinessParking'" or "{ 'BusinessParking'" (i.e., with or without curly braces), which highlights the need for careful parsing and structuring of the data during cleaning and transformation.

A Python script can be used to perform all of these actions, including transforming and cleaning the data, before uploading it into a structured database.

5 General Comments

We acknowledge that our ER model could potentially be further optimized for performance. However, in addition to optimizing performance, we also aimed to keep the model clear and user-friendly. Thus, we believe that our proposed model strikes a good balance between these two aspects.

Through the process of designing this model, we gained a deeper understanding of the challenges that database engineers may face, such as dealing with unorganized and inconsistent data files. These are all the challenges that an engineer should be prepare to deal with.

6 Annex

This rubric give all the figure that might need a bigger impression to clearly understand it.

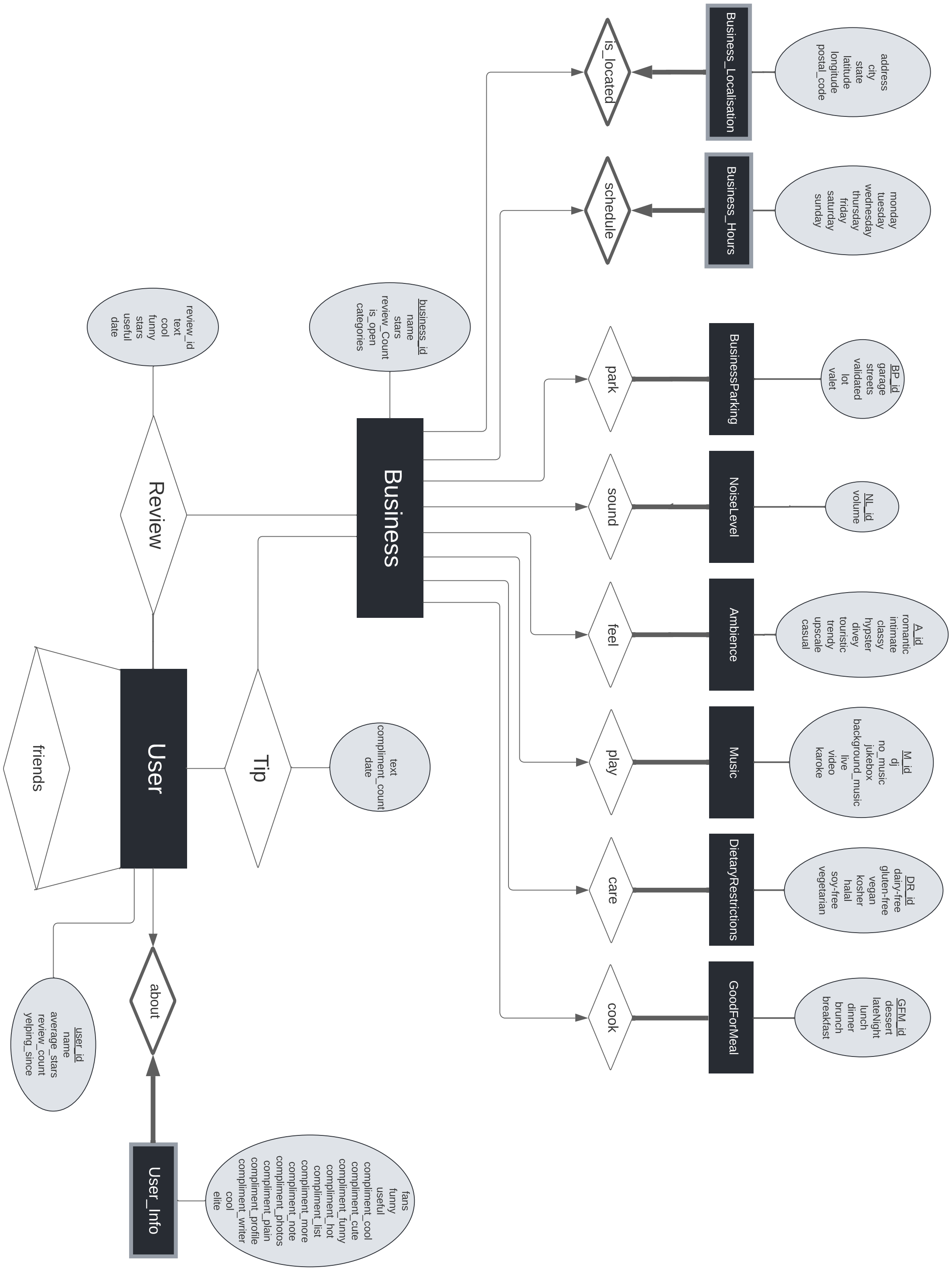


Figure 8: ER model simplified (large)

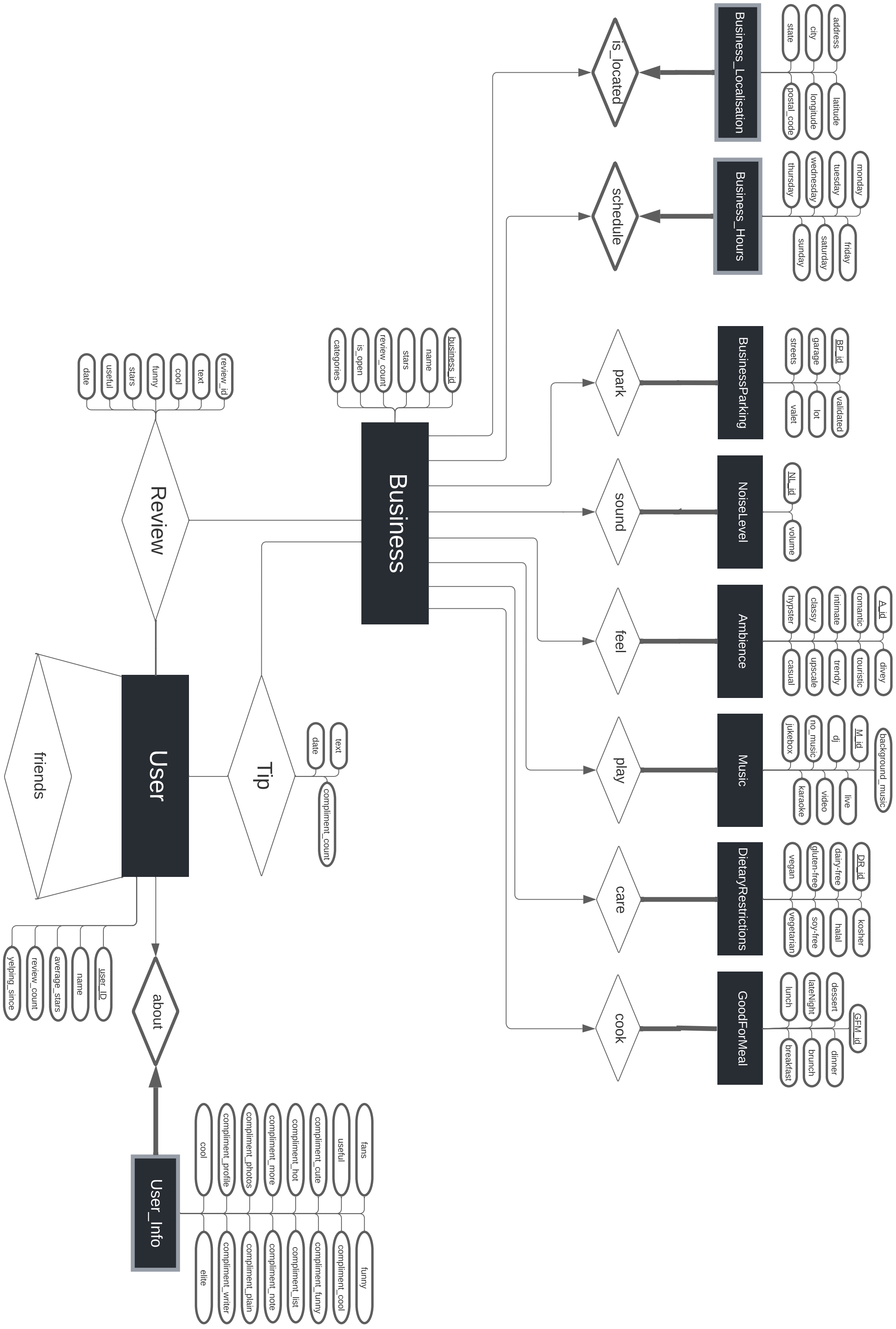
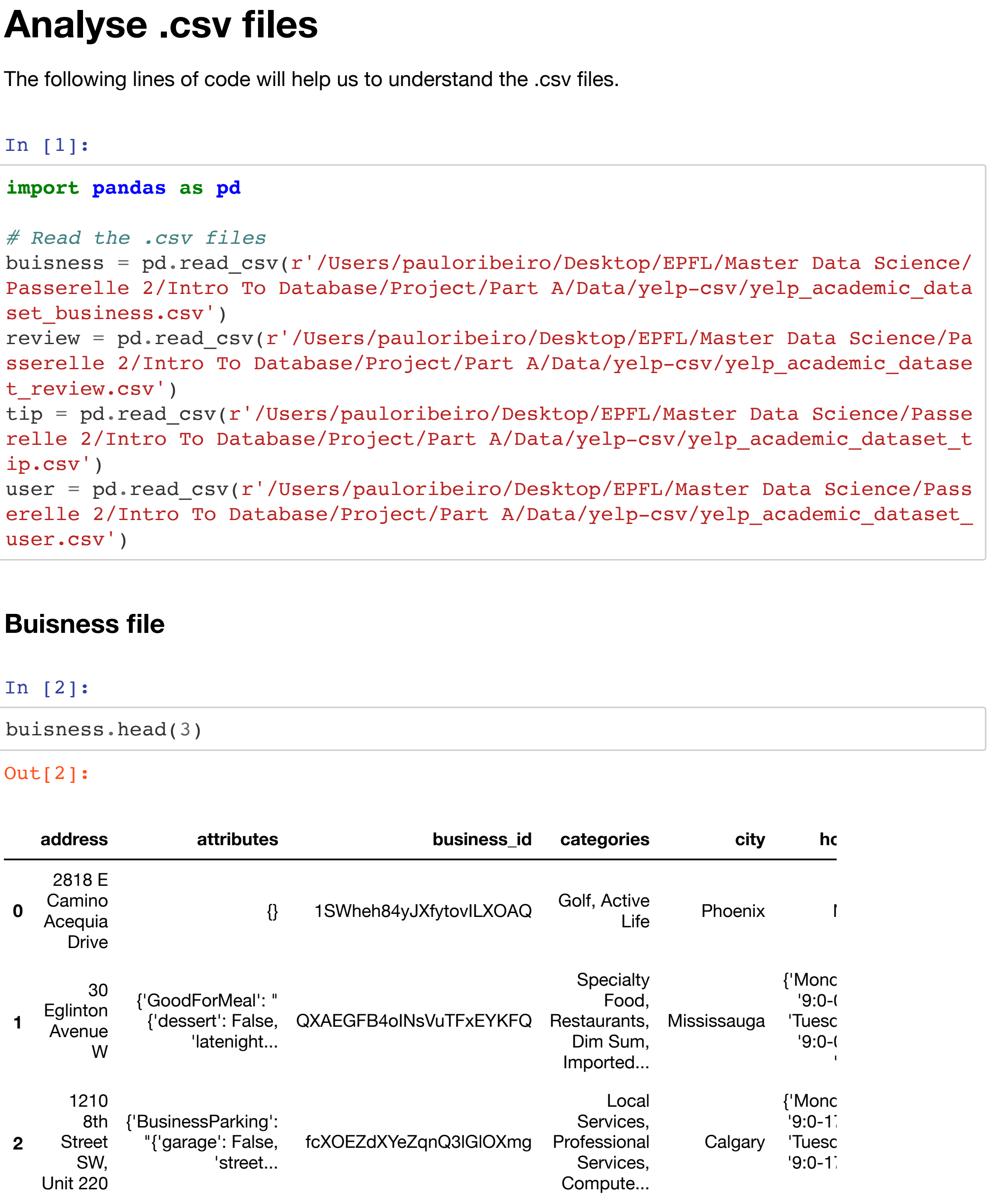


Figure 9: ER model (large)

7 Python script

The Python script was instrumental in comprehending and examining the data, allowing us to formulate our assumptions and build our model accordingly. Here is our script.



Analyse .csv files

The following lines of code will help us to understand the .csv files.

In [1]:

```
import pandas as pd

# Read the .csv files
buisness = pd.read_csv(r'/Users/pauloribeiro/Desktop/EPFL/Master Data Science/
Passerelle 2/Intro To Database/Project/Part A/Data/yelp-csv/yelp_academic_data
set_business.csv')
review = pd.read_csv(r'/Users/pauloribeiro/Desktop/EPFL/Master Data Science/Pa
sserelle 2/Intro To Database/Project/Part A/Data/yelp-csv/yelp_academic_datase
t_review.csv')
tip = pd.read_csv(r'/Users/pauloribeiro/Desktop/EPFL/Master Data Science/Passe
relle 2/Intro To Database/Project/Part A/Data/yelp-csv/yelp_academic_dataset_t
ip.csv')
user = pd.read_csv(r'/Users/pauloribeiro/Desktop/EPFL/Master Data Science/Pass
erelle 2/Intro To Database/Project/Part A/Data/yelp-csv/yelp_academic_dataset_
user.csv')
```

Buisness file

In [2]:

```
buisness.head(3)
```

Out[2]:

	address	attributes	business_id	categories	city	hc
0	2818 E Camino Acequia Drive	{}	1SWheh84yJXfytovILXOAQ	Golf, Active Life	Phoenix	N
1	30 Eglinton Avenue W	{'GoodForMeal': "{'dessert': False, 'latenight...	QXAEGFB4oINsVuTFxEYKFQ	Specialty Food, Restaurants, Dim Sum, Imported...	Mississauga	{'Monc '9:0-('Tuesc '9:0-('
2	1210 8th Street SW, Unit 220	{'BusinessParking': "{'garage': False, 'street...	fcXOEZdXYeZqnQ3lGlOXmg	Local Services, Professional Services, Compute...	Calgary	{'Monc '9:0-1 'Tuesc '9:0-1

In [3]:

```
print(f'Number of row : {buisness.shape[0]}')
```

```
Number of row : 192609
```

Review file

In [4]:

```
review.head(3)
```

Out[4]:

	business_id	cool	date	funny	review_id	stars	
0	WTqjgwHlXbSFevF32_DJVw	0	2016-11-09 20:09:03	0.0	2TzJjDVDEuAW6MR5Vuc1ug	5.0	I ha sa has
1	b1b1eb3uo-w561D0ZfCEiQ	0	2018-01-30 23:07:38	0.0	11a8sVPMUFtaC7_ABRkmtw	1.0	w se ses
2	3fw2X5bZYeW9xCz_zGhOHg	5	2016-05-07 01:21:02	4.0	G7XHMxG0bx9oBJNECG4IFg	3.0	d na a

In [5]:

```
print(f'Number of row : {review.shape[0]}')
```

```
Number of row : 918680
```

Tip file

```
In [6]:
```

```
tip.head(3)
```

```
Out[6]:
```

	business_id	compliment_count	date	text	
0	VaKXUpmWTTWDKbpJ3aQdMw	0	2014-03-27 03:51:24	Great for watching games, ufc, and whatever el...	UPw5DWs_b-e2JRB
1	OPiPeoJiv92rENwbq76orA	0	2013-05-25 06:00:56	Happy Hour 2-4 daily with 1/2 price drinks and...	Ocha4kZBHb4JK0lO
2	5KheTjYPu1HcQzQFtm4_vw	0	2011-12-26 01:46:17	Good chips and salsa. Loud at times. Good serv...	jRyO2V1pA4CdVVqC

```
In [7]:
```

```
print(f'Number of row : {tip.shape[0]}')
```

```
Number of row : 1029047
```

User file

In [8]:

```
user.head(3)
```

Out[8]:

	average_stars	compliment_cool	compliment_cute	compliment_funny	compliment_hot	c
0	4.03	1	0	1	2	
1	3.63	1	0	1	1	
2	3.71	0	0	0	0	

3 rows × 22 columns

In [9]:

```
print(f'Number of row : {user.shape[0]}')
```

```
Number of row : 778651
```

Analyze the Data

Now that we have better seen and understood the construction of the data, we can analyze it. Let's have a look first at the buisness' attribute named "attributes", "address" and "friends".

In [10]:

```
curr_set = buisness
look_at = input('Enter the name of the attribute in your current set that you
are interested for:')
atts = curr_set[look_at]

#Here we go through the data to check the types of it.
types = []
for att in atts:
    if type(att) in types:
        pass
    else:
        types.append(type(att))

#We see that in our data we have two types of data given. It will be our missi
on to clean that
#and return only one type. (DATA CLEANING)
print(f"Different types of data in {look_at} : \n", types, "\n")

#Let's clean the words (remove all the ", ', {, ...)
def CleanWord(DirtyWord: str):
    return DirtyWord.translate({ord(i): None for i in '"{:}'})

# We have seen that all attributes begin with a capital. So we extract all of
them with this "cheat".
all_att = []
for att in atts:
    if type(att) == str :
        for word in att.split():
            for i in range(len(word)):
                if word[i].isupper() and not("False" in word) and not("True" i
n word) and not("None" in word):
                    if CleanWord(word) in all_att:
                        pass
                    else:
                        all_att.append(CleanWord(word))

print(f'All sub-attributes that appears in {look_at} ({len(all_att)}): \n', al
l_att,)
```

```
Different types of data in attributes :
 [<class 'str'>, <class 'float'>]

All sub-attributes that appears in attributes (6):
 ["'GoodForMeal'", "'BusinessParking'", "'NoiseLevel'", "'Ambience
'", "'Music'", "'DietaryRestrictions'"]
```

In [11]:

```
#Prove that not every businesses have an address recorded.
atts = buisness["address"]

for att in atts:
    if type(att) == float:
        print(True, f"-> record found : {att}", "\nNot every business has its
adress in the Yelp! dataset.")
        break
```

```
True -> record found : nan
Not every business has its adress in the Yelp! dataset.
```

In [12]:

```
#Prove that not every businesses have an "attributes" values recorded.
atts = buisness["attributes"]

for att in atts:
    if type(att) == float:
        print(True, f"-> record found : {att}", "\nNot every business has its
attributes in the Yelp! dataset.")
        break
```

```
True -> record found : nan
Not every business has its attributes in the Yelp! dataset.
```

In [13]:

```
curr_set = user
look_at = input('Enter the name of the attribute in your current set that you
are interested for:')
atts = curr_set[look_at]
stop = 5 #print the 5 first user with no friends.

for i in range(len(atts)):
    if len(atts[i])>0 and len(atts[i])<26: #26 is the length of someone with o
ne friend.
        stop -= 1
        print(atts[i], i)
        print(user["user_id"][i])
        print("\n")

        if stop == 0 : #allow to stop the loop after 5 prints.
            break
    else:
        pass

if stop == 5: #means that no one has no friend
    print("Everyone is friend with at least one person")
```

```
Everyone is friend with at least one person
```