DIAS: Data-Intensive Applications and Systems Laboratory
School of Computer and Communication Sciences
Ecole Polytechnique Fédérale de Lausanne
Building BC, Station 14
CH-1015 Lausanne
URL: http://dias.epfl.ch/

Databases Project – Spring 2020

Team No: 04

Members: Mateo Chapatte, Thomas Wilde, Guillaume Parchet

Contents

DIAS: Data-Intensive Applications and Systems Laboratory
School of Computer and Communication Sciences
Ecole Polytechnique Fédérale de Lausanne
Building BC, Station 14
CH-1015 Lausanne
URL: http://dias.epfl.ch/

Deliverable 1

Assumptions

The assumptions we made on the input data:

- A common address has 100 or less characters.
- A common city has 35 or less characters.
- All businesses have and provide a unique business id consisting of 22 characters.
- The name of a business has 60 or less characters.
- A postal code has 12 or less characters (Oracle's recommendation).
- Latitude and longitudes have a maximum precision of 7 digits after comma and their absolute value is below 1000.
- Stars' value is between 0 and 5 and has precision of one digit after the decimal point.
- States' initials can be written with 2 or less characters.
- A noise level has only three possible values: quiet, average or loud.
- A user has a name that is 45 or less characters.
- All users have and provide a unique user id consisting of 22 characters.
- All users have a distinct id consisting of 22 characters.
- A review text is truncated after a maximum of 50 characters

We assume a User can add multiple Reviews of a given business (as there is a unique review id).

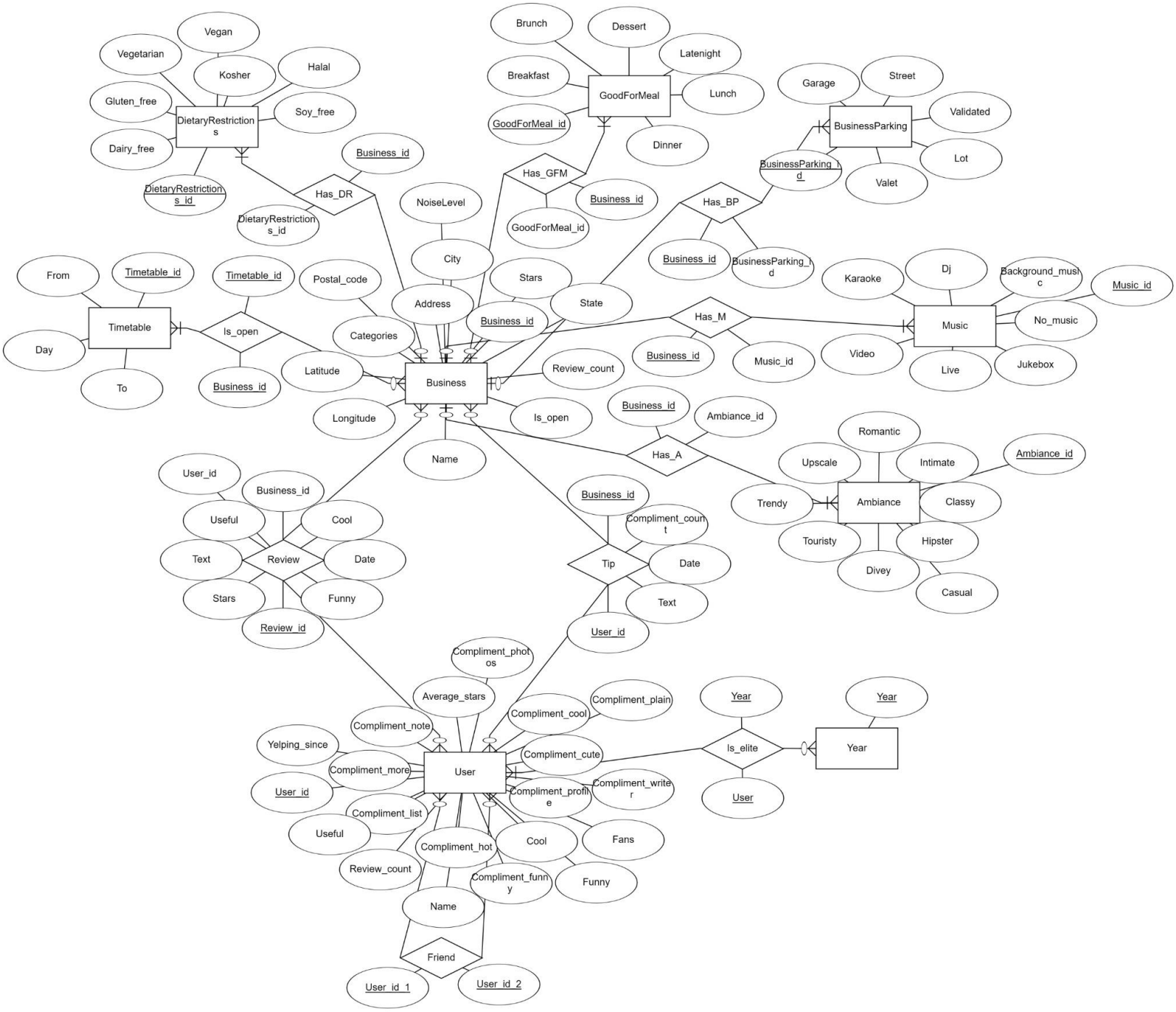
We assume a User can only publish one tip for a given business (since tip has no id, we decided to user the combination of business id and user id as its primary key).

All friendship relations are symmetric (we checked on the yelp platform and, in the dataset, it was the case for both).

We chose to represent ids as strings of 22 characters as we checked it is the format chosen in the yelp dataset.

DIAS: Data-Intensive Applications and Systems Laboratory
School of Computer and Communication Sciences
Ecole Polytechnique Fédérale de Lausanne
Building BC, Station 14
CH-1015 Lausanne
URL: http://dias.epfl.ch/

Entity Relationship Schema



DIAS: Data-Intensive Applications and Systems Laboratory
School of Computer and Communication Sciences
Ecole Polytechnique Fédérale de Lausanne
Building BC, Station 14
CH-1015 Lausanne
URL: http://dias.epfl.ch/

Description

It seemed natural to have two main entities business and user since they represent real world objects. We also added some extra entities for the following reasons:

- The field “Attributes” has 6 specializations. Since they are all represented by a tuple of Booleans (expect for Noise Level, we used an enumeration since it only has 3 possible values), we decided to create a table corresponding to each of these specializations. This allows to only create one instance of a specialization tuple when multiple businesses share the same one. Also, it helps to model that a business might not have all / any of the “Attributes” but every specialization has to be mapped onto a corresponding business. As there is a maximum of 512 combinations for specialization tuples and about hundreds of thousands of businesses, we save space by mapping businesses onto some standardized tuples.
- For similar reasons, we created a Timetable entity (corresponding to “Hours” field). It permits to share common opening hours to multiple businesses (some are very standard).
- The Year entity is linked to the “Elite” field. We decided not to model it as a String listing the elite years since it complicates some searches (e.g. to find all users that were elites on a range of years) so we preferred to map a user to a year he has been elite. Modeling it as an integer allows to perform mathematical expressions on elite years.

The Tip and Review relations between Business and Users also came naturally, we decided to add an extra one to model the friendship relation between two users.

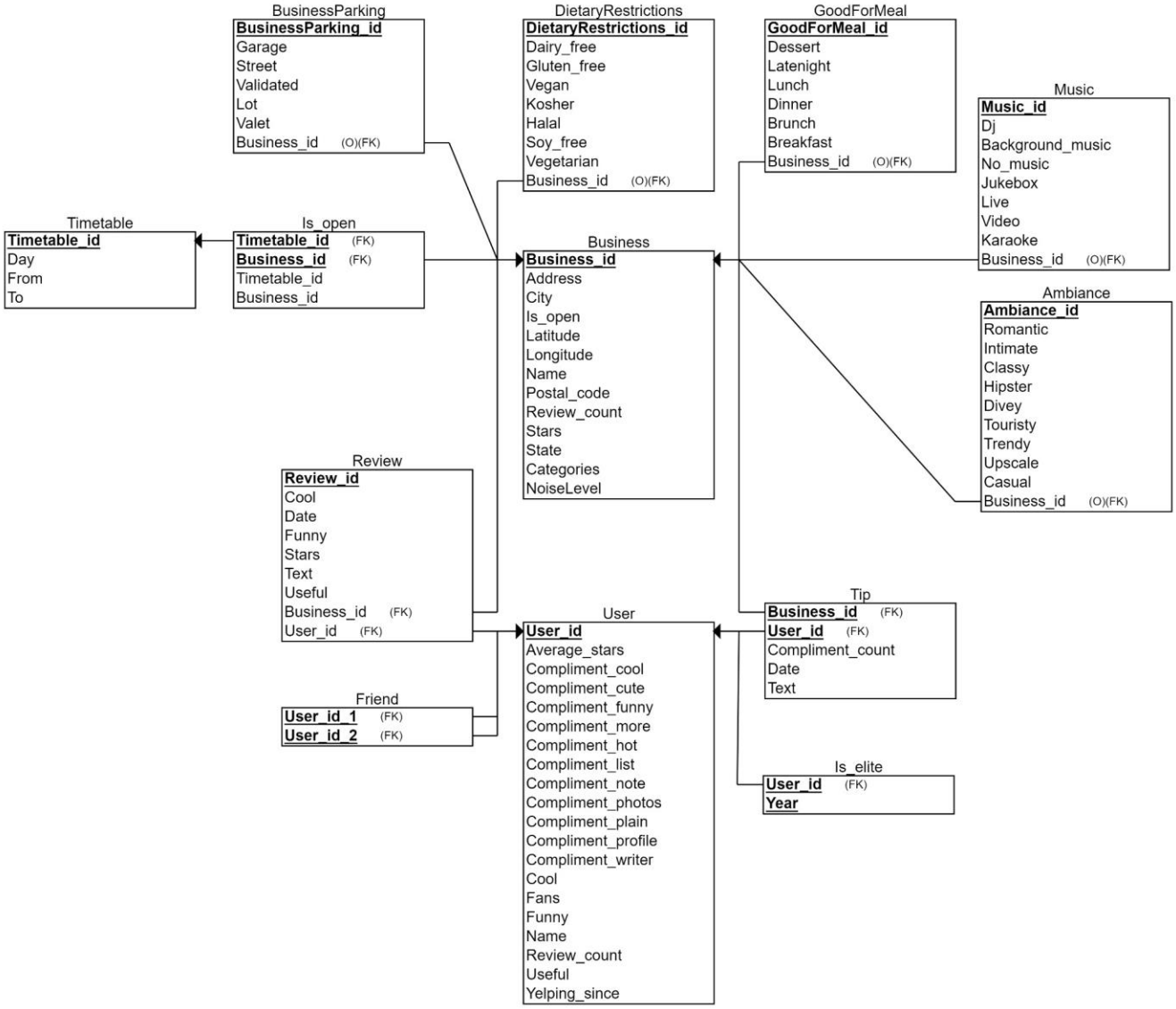
Comment: shall we add more precisions? For example to come back on the assumptions choices we made or is it redundant?

DIAS: Data-Intensive Applications and Systems Laboratory
School of Computer and Communication Sciences
Ecole Polytechnique Fédérale de Lausanne
Building BC, Station 14
CH-1015 Lausanne
URL: http://dias.epfl.ch/

Relational Schema

ER schema to Relational schema

<Describe the transition from ER schema to Relational schema>



DIAS: Data-Intensive Applications and Systems Laboratory
School of Computer and Communication Sciences
Ecole Polytechnique Fédérale de Lausanne
Building BC, Station 14
CH-1015 Lausanne
URL: http://dias.epfl.ch/

DDL

```
CREATE TABLE Business
(
 Address CHAR(100),
 Business_id CHAR(22) NOT NULL,
 City CHAR(35),
 Is_open BOOLEAN,
 Latitude DOUBLE(10,7),
 Longitude DOUBLE(10,7),
 Name CHAR(60),
 Postal_code CHAR(12),
 Review_count INT UNSIGNED,
 Stars DOUBLE(2,1) UNSIGNED,
 State CHAR(2),
 Categories VARCHAR,
 NoiseLevel ENUM('quiet','average','loud'),
 PRIMARY KEY (Business_id)
);

CREATE TABLE User
(
 Average_stars DOUBLE(3,2),
 Compliment_cool INT UNSIGNED,
 Compliment_cute INT UNSIGNED,
 Compliment_funny INT UNSIGNED,
 Compliment_more INT UNSIGNED,
 Compliment_hot INT UNSIGNED,
 Compliment_list INT UNSIGNED,
 Compliment_note INT UNSIGNED,
 Compliment_photos INT UNSIGNED,
 Compliment_plain INT UNSIGNED,
 Compliment_profile INT UNSIGNED,
 Compliment_writer INT UNSIGNED,
 Cool INT UNSIGNED,
 Fans INT UNSIGNED,
 Funny INT UNSIGNED,
 Name CHAR(45),
 Review_count INT UNSIGNED,
 Useful INT UNSIGNED,
 User_id CHAR(22) NOT NULL,
 Yelping_since DATETIME,
 PRIMARY KEY (User_id)
);

CREATE TABLE GoodForMeal
(
```

DIAS: Data-Intensive Applications and Systems Laboratory
School of Computer and Communication Sciences
Ecole Polytechnique Fédérale de Lausanne
Building BC, Station 14
CH-1015 Lausanne
URL: http://dias.epfl.ch/

```
 Dessert BOOLEAN NOT NULL,
 Latenight BOOLEAN NOT NULL,
 Lunch BOOLEAN NOT NULL,
 Dinner BOOLEAN NOT NULL,
 Brunch BOOLEAN NOT NULL,
 Breakfast BOOLEAN NOT NULL,
 GoodForMeal_id TINYINT UNSIGNED NOT NULL AUTO_INCREMENT,
 PRIMARY KEY (GoodForMeal_id)
);

CREATE TABLE Has_GFM
(
 Business_id CHAR(22) NOT NULL,
 GoodForMeal_id TINYINT UNSIGNED NOT NULL,
 FOREIGN KEY (Business_id) REFERENCES Business(Business_id),
 FOREIGN KEY (GoodForMeal_id) REFERENCES GoodForMeal(GoodForMeal_id),
 PRIMARY KEY (Business_id)
);

CREATE TABLE BusinessParking
(
 Garage BOOLEAN NOT NULL,
 Street BOOLEAN NOT NULL,
 Validated BOOLEAN NOT NULL,
 Lot BOOLEAN NOT NULL,
 Valet BOOLEAN NOT NULL,
 BusinessParking_id TINYINT UNSIGNED NOT NULL AUTO_INCREMENT,
 PRIMARY KEY (BusinessParking_id)
);

CREATE TABLE Has_BP
(
 Business_id CHAR(22) NOT NULL,
 BusinessParking_id TINYINT UNSIGNED NOT NULL,
 FOREIGN KEY (Business_id) REFERENCES Business(Business_id),
 FOREIGN KEY (BusinessParking_id) REFERENCES BusinessParking(BusinessParking_id),
 PRIMARY KEY (Business_id)
);

CREATE TABLE Music
(
 Dj BOOLEAN NOT NULL,
 Background_music BOOLEAN NOT NULL,
 No_music BOOLEAN NOT NULL,
 Jukebox BOOLEAN NOT NULL,
 Live BOOLEAN NOT NULL,
 Video BOOLEAN NOT NULL,
```

DIAS: Data-Intensive Applications and Systems Laboratory
School of Computer and Communication Sciences
Ecole Polytechnique Fédérale de Lausanne
Building BC, Station 14
CH-1015 Lausanne
URL: http://dias.epfl.ch/

```
 Karaoke BOOLEAN NOT NULL,
 Music_id TINYINT UNSIGNED NOT NULL AUTO_INCREMENT,
 PRIMARY KEY (Music_id)
);

CREATE TABLE Has_M
(
 Business_id CHAR(22) NOT NULL,
 Music_id TINYINT UNSIGNED NOT NULL,
 FOREIGN KEY (Business_id) REFERENCES Business(Business_id),
 FOREIGN KEY (Music_id) REFERENCES Music(Music_id),
 PRIMARY KEY (Business_id)
);


CREATE TABLE Ambiance
(
 Romantic BOOLEAN NOT NULL,
 Intimate BOOLEAN NOT NULL,
 Classy BOOLEAN NOT NULL,
 Hipster BOOLEAN NOT NULL,
 Divey BOOLEAN NOT NULL,
 Touristy BOOLEAN NOT NULL,
 Trendy BOOLEAN NOT NULL,
 Upscale BOOLEAN NOT NULL,
 Casual BOOLEAN NOT NULL,
 Ambiance_id TINYINT UNSIGNED NOT NULL AUTO_INCREMENT,
 PRIMARY KEY (Ambiance_id)
);

CREATE TABLE Has_A
(
 Business_id CHAR(22) NOT NULL,
 Ambiance_id TINYINT UNSIGNED NOT NULL,
 FOREIGN KEY (Business_id) REFERENCES Business(Business_id),
 FOREIGN KEY (Ambiance_id) REFERENCES Ambiance(Ambiance_id),
 PRIMARY KEY (Business_id)
);

CREATE TABLE DietaryRestrictions
(
 Dairy_free BOOLEAN NOT NULL,
 Gluten_free BOOLEAN NOT NULL,
 Vegan BOOLEAN NOT NULL,
 Kosher BOOLEAN NOT NULL,
 Halal BOOLEAN NOT NULL,
 Soy_free BOOLEAN NOT NULL,
```

DIAS: Data-Intensive Applications and Systems Laboratory
School of Computer and Communication Sciences
Ecole Polytechnique Fédérale de Lausanne
Building BC, Station 14
CH-1015 Lausanne
URL: http://dias.epfl.ch/

```
 Vegetarian BOOLEAN NOT NULL,
 DietaryRestrictions_id TINYINT UNSIGNED NOT NULL AUTO_INCREMENT,
 PRIMARY KEY (DietaryRestrictions_id)
);

CREATE TABLE Has_DR
(
 Business_id CHAR(22) NOT NULL,
 DietaryRestrictions_id TINYINT UNSIGNED NOT NULL,
 FOREIGN KEY (Business_id) REFERENCES Business(Business_id),
 FOREIGN KEY (DietaryRestrictions_id) REFERENCES DietaryRestrictions(DietaryRestrictions_id),
 PRIMARY KEY (Business_id)
);

CREATE TABLE Timetable
(
 Day ENUM('Monday','Tuesday','Wednesday','Thursday','Friday','Saturday','Sunday') NOT NULL,
 From TIME NOT NULL,
 To TIME NOT NULL,
 Timetable_id INT UNSIGNED NOT NULL AUTO_INCREMENT,
 PRIMARY KEY (Timetable_id)
);

CREATE TABLE Is_open
(
 Timetable_id INT UNSIGNED NOT NULL,
 Business_id CHAR(22) NOT NULL,
 PRIMARY KEY (Timetable_id, Business_id),
 FOREIGN KEY (Timetable_id) REFERENCES Timetable(Timetable_id),
 FOREIGN KEY (Business_id) REFERENCES Business(Business_id)
);

CREATE TABLE Review
(
 Cool INT UNSIGNED,
 Date DATETIME,
 Funny INT UNSIGNED,
 Review_id CHAR(22) NOT NULL,
 Stars INT UNSIGNED,
 Text CHAR(50),
 Useful INT UNSIGNED,
 User_id CHAR(22) NOT NULL,
 Business_id CHAR(22) NOT NULL,
 PRIMARY KEY (Review_id),
 FOREIGN KEY (User_id) REFERENCES User(User_id),
 FOREIGN KEY (Business_id) REFERENCES Business(Business_id)
);
```

DIAS: Data-Intensive Applications and Systems Laboratory
School of Computer and Communication Sciences
Ecole Polytechnique Fédérale de Lausanne
Building BC, Station 14
CH-1015 Lausanne
URL: http://dias.epfl.ch/

```
CREATE TABLE Tip
(
 Compliment_count INT UNSIGNED,
 Date DATETIME,
 Text VARCHAR,
 User_id CHAR(22) NOT NULL,
 Business_id CHAR(22) NOT NULL,
 PRIMARY KEY (User_id, Business_id),
 FOREIGN KEY (User_id) REFERENCES User(User_id),
 FOREIGN KEY (Business_id) REFERENCES Business(Business_id)
);

CREATE TABLE Friend
(
 User_id_1 CHAR(22) NOT NULL,
 User_id_2 CHAR(22) NOT NULL,
 PRIMARY KEY (User_id_1, User_id_2),
 FOREIGN KEY (User_id_1) REFERENCES User(User_id),
 FOREIGN KEY (User_id_2) REFERENCES User(User_id)
);

CREATE TABLE Is_elite
(
 Year YEAR NOT NULL,
 User_id CHAR(22) NOT NULL,
 PRIMARY KEY (Year, User_id),
 FOREIGN KEY (User_id) REFERENCES User(User_id)
);
```

General Comments

We mostly worked together (using Discord since the epidemic broke out of course) using screen sharing.

DIAS: Data-Intensive Applications and Systems Laboratory
School of Computer and Communication Sciences
Ecole Polytechnique Fédérale de Lausanne
Building BC, Station 14
CH-1015 Lausanne
URL: http://dias.epfl.ch/

Deliverable 2

Assumptions

We added some changes from Milestone 1:

We removed the keys that should not have been on the ER model

ER model has new tables for Locations (City, Postal_code + relations)

We decided to use Oracle Database so we needed to adapt most of the types of our DDL statements to make it compatible with the database.

For some attributes like city and address we found valid data a little too long, we though truncating was not the most adapted for these kind of attributes so we extended a little the maximum number of characters, we felt like some of our limit were a little too low.

We remarked some texts / tips / reviews /… had characters that were not in standard UTF8 so we extended further the maximum size of some CHAR attributes and truncated the others so they fit in the requirements (text in Tips, text in Review, …).

While checking the data, we realized users may have multiple tips concerning a single business. We changed the key to an extended tuple (business_id, user_id, date) as a user won't publish two tips at the very same time.

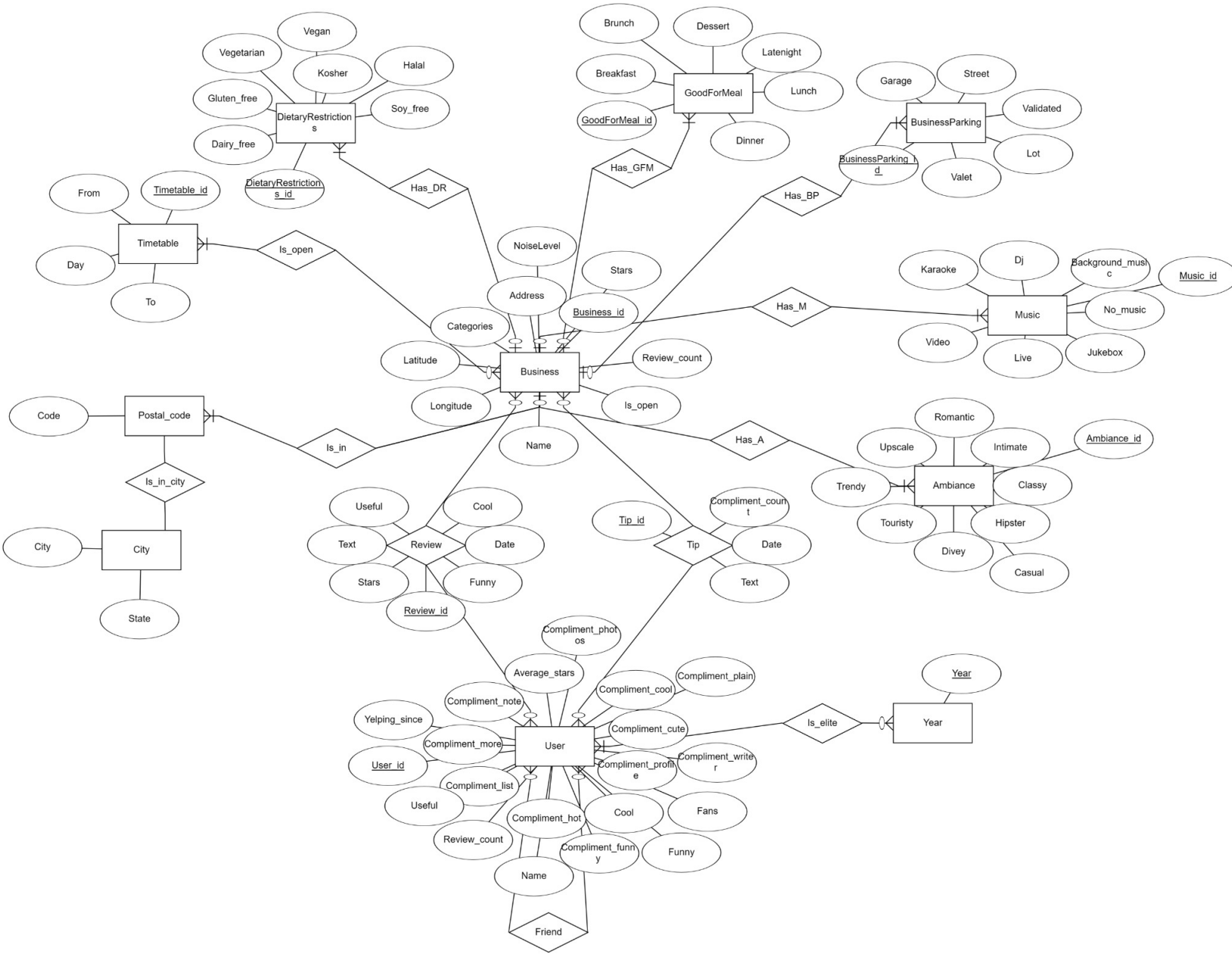
We decided to sanitize a little the names of the cities. We've set them all to lower case, dropped the stress (accents) and stripped the extra spaces after the city's name.

Those changes changed the ER relational schema, ER table and the DDL statements. You can find the updated versions below.

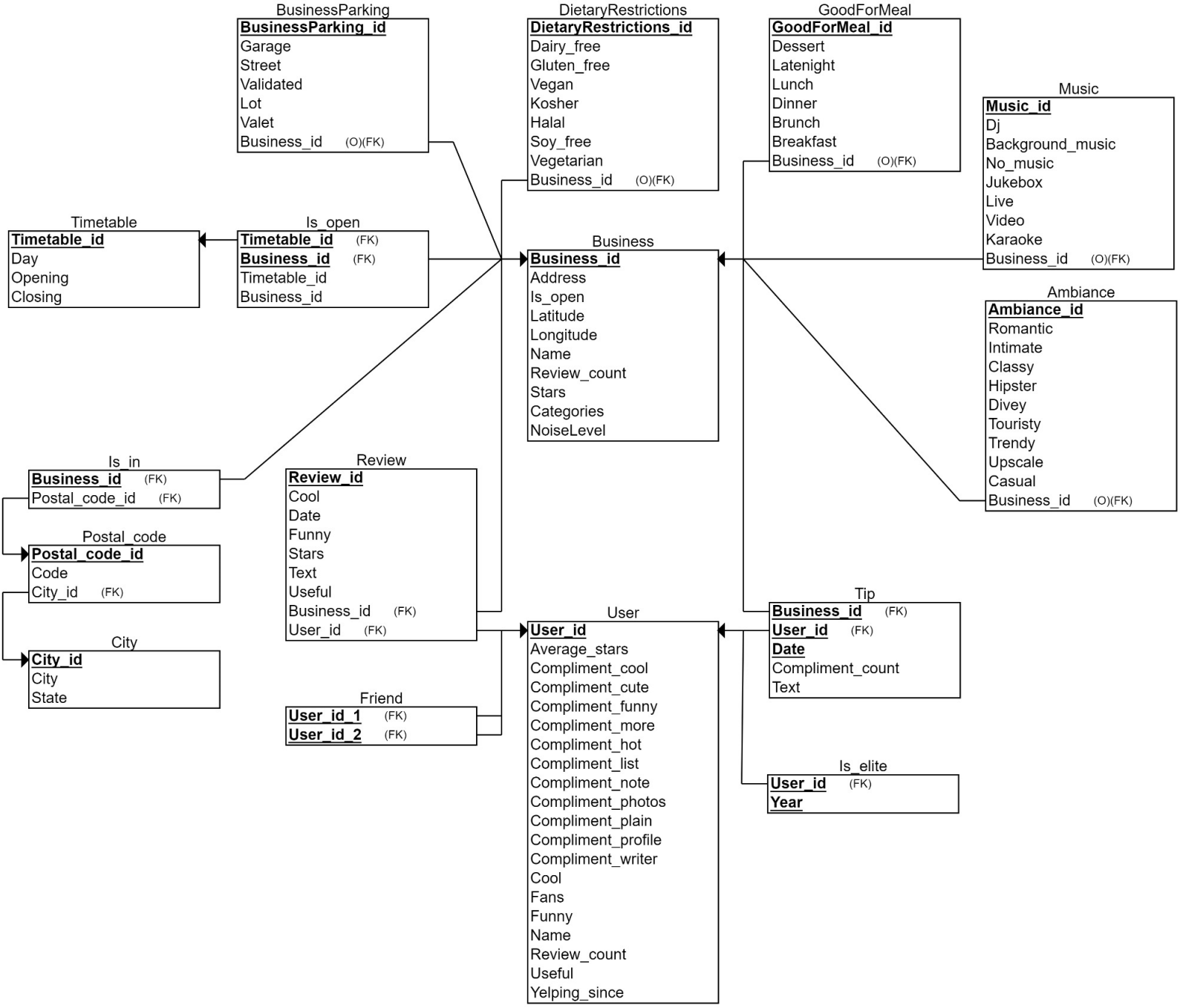
We modified the following assumptions on the input data (compared to milestone 1):

- A common address has 120 or less characters.
- A common city has 50 or less characters.
- The name of a business has 65 or less characters.
- Latitude has absolute value below 100.
- States' initials can be written with 3 or less characters.
- A noise level has only four possible values: quiet, average, loud, very_loud.
- A review / tip text is truncated after a maximum of 50 characters encoded in UTF-8 encoding
- A User can publish multiple tips for a given business but not at the very same time (same second).
- We keep the given yelp ids as strings of 22 characters but generate integer ids for the attributes that need one but had none given by the dataset.

DIAS: Data-Intensive Applications and Systems Laboratory
School of Computer and Communication Sciences
Ecole Polytechnique Fédérale de Lausanne
Building BC, Station 14
CH-1015 Lausanne
URL: http://dias.epfl.ch/



DIAS: Data-Intensive Applications and Systems Laboratory
School of Computer and Communication Sciences
Ecole Polytechnique Fédérale de Lausanne
Building BC, Station 14
CH-1015 Lausanne
URL: http://dias.epfl.ch/



Data Loading/Cleaning

Loading the Data:

To load the data into the new tables we used our newly created cleaned data .csv files

We changed the DDL statements, adapted them for Oracle database (multiple types we though using during milestone 1 were not supported):

```
CREATE TABLE Business
(
 Address VARCHAR2(120),
 Business_id CHAR(22) NOT NULL,
 Is_open NUMBER(1) CHECK (Is_open IN (0,1)),
 Latitude NUMBER(9,7),
 Longitude NUMBER(10,7),
 Name VARCHAR2(65),
 Review_count NUMBER(10) CHECK (Review_count >=0),
 Stars NUMBER(2,1) CHECK (Stars >=0),
 Categories VARCHAR2(600),
 NoiseLevel VARCHAR2(10) CHECK(NoiseLevel IN ('quiet','average','loud','very_loud')),
 PRIMARY KEY (Business_id)
);

CREATE TABLE USERS
(
 Average_stars NUMBER(3,2) CHECK (Average_stars >=0),
 Compliment_cool INT CHECK (Compliment_cool >=0),
 Compliment_cute INT CHECK (Compliment_cute >=0),
 Compliment_funny INT CHECK (Compliment_funny >=0),
 Compliment_more INT CHECK (Compliment_more >=0),
 Compliment_hot INT CHECK (Compliment_hot >=0),
 Compliment_list INT CHECK (Compliment_list >=0),
 Compliment_note INT CHECK (Compliment_note >=0),
 Compliment_photos INT CHECK (Compliment_photos >=0),
 Compliment_plain INT CHECK (Compliment_plain >=0),
 Compliment_profile INT CHECK (Compliment_profile >=0),
 Compliment_writer INT CHECK (Compliment_writer >=0),
 Cool INT CHECK (Cool >=0),
 Fans INT CHECK (Fans >=0),
 Funny INT CHECK (Funny >=0),
 Name VARCHAR2(45),
 Review_count INT CHECK (Review_count >=0),
 Useful INT CHECK (Useful >=0),
 User_id CHAR(22) NOT NULL,
 Yelping_since TIMESTAMP(0),
 PRIMARY KEY (User_id)
);
```

```
CREATE TABLE Review
(
 Cool INT CHECK(cool >=0),
 Date_of TIMESTAMP(0),
 Funny INT CHECK(Funny >=0),
 Review_id CHAR(22) NOT NULL,
 Stars INT CHECK(Stars >=0),
 Text VARCHAR2(50),
 Useful INT CHECK(Useful >=0),
 User_id CHAR(22) NOT NULL,
 Business_id CHAR(22) NOT NULL,
 PRIMARY KEY (Review_id),
 FOREIGN KEY (User_id) REFERENCES USERS(User_id),
 FOREIGN KEY (Business_id) REFERENCES BUSINESS(Business_id)
);

CREATE TABLE Tip
(
 Compliment_count INT CHECK(Compliment_count >=0),
 Dated TIMESTAMP(0) NOT NULL,
 Text VARCHAR2(50),
 User_id CHAR(22) NOT NULL,
 Business_id CHAR(22) NOT NULL,
 PRIMARY KEY (Business_id, User_id, Dated),
 FOREIGN KEY (User_id) REFERENCES Users(User_id),
 FOREIGN KEY (Business_id) REFERENCES Business(Business_id)
);

CREATE TABLE Friend
(
 User_id_1 CHAR(22) NOT NULL,
 User_id_2 CHAR(22) NOT NULL,
 PRIMARY KEY (User_id_1, User_id_2),
 FOREIGN KEY (User_id_1) REFERENCES Users(User_id),
 FOREIGN KEY (User_id_2) REFERENCES Users(User_id)
);

CREATE TABLE IS_ELITE
(
 year NUMBER(4) NOT NULL,
 User_id CHAR(22) NOT NULL,
 PRIMARY KEY (Year, User_id),
 FOREIGN KEY (User_id) REFERENCES Users(User_id)
);

CREATE TABLE Postal_code
(
 Code VARCHAR2(12),
 City_id NUMBER NOT NULL,
 Postal_code_id NUMBER GENERATED ALWAYS as IDENTITY(START with 1 INCREMENT by 1),
```

```
 PRIMARY KEY (Postal_code_id)
);

CREATE TABLE City
(
 City VARCHAR2(50),
 State VARCHAR2(3),
 City_id NUMBER GENERATED ALWAYS as IDENTITY(START with 1 INCREMENT by 1),
 PRIMARY KEY (City_id)
);

CREATE TABLE Is_in
(
 Business_id CHAR(22) NOT NULL,
 Postal_code_id NUMBER NOT NULL,
 PRIMARY KEY (Business_id),
 FOREIGN KEY (Business_id) REFERENCES Business(Business_id),
 FOREIGN KEY (Postal_code_id) REFERENCES Postal_code(Postal_code_id)
);

CREATE TABLE Timetable
(
 Day VARCHAR2(12) CHECK(Day IN('Monday','Tuesday','Wednesday','Thursday','Friday','Saturday','Sunday'))
NOT NULL,
 Opening DATE NOT NULL,
 Closing DATE NOT NULL,
 Timetable_id NUMBER GENERATED ALWAYS as IDENTITY(START with 1 INCREMENT by 1),
 PRIMARY KEY (Timetable_id)
);

CREATE TABLE Is_open
(
 Timetable_id NUMBER NOT NULL,
 Business_id CHAR(22) NOT NULL,
 PRIMARY KEY (Timetable_id, Business_id),
 FOREIGN KEY (Timetable_id) REFERENCES Timetable(Timetable_id),
 FOREIGN KEY (Business_id) REFERENCES Business(Business_id)
);

CREATE TABLE GoodForMeal
(
 Dessert NUMBER(1) CHECK (Dessert IN (0,1)) NOT NULL,
 Latenight NUMBER(1) CHECK (Latenight IN (0,1)) NOT NULL,
 Lunch NUMBER(1) CHECK (Lunch IN (0,1)) NOT NULL,
 Dinner NUMBER(1) CHECK (Dinner IN (0,1)) NOT NULL,
 Brunch NUMBER(1) CHECK (Brunch IN (0,1)) NOT NULL,
 Breakfast NUMBER(1) CHECK (Breakfast IN (0,1)) NOT NULL,
 GoodForMeal_id NUMBER(3) GENERATED ALWAYS as IDENTITY(START with 1 INCREMENT by 1),
 PRIMARY KEY (GoodForMeal_id)
);
```

```
CREATE TABLE Has_GFM
(
 Business_id CHAR(22) NOT NULL,
 GoodForMeal_id NUMBER(3) NOT NULL,
 FOREIGN KEY (Business_id) REFERENCES Business(Business_id),
 FOREIGN KEY (GoodForMeal_id) REFERENCES GoodForMeal(GoodForMeal_id),
 PRIMARY KEY (Business_id)
);

CREATE TABLE BusinessParking
(
 Garage NUMBER(1) CHECK (Garage IN (0,1)) NOT NULL,
 Street NUMBER(1) CHECK (Street IN (0,1)) NOT NULL,
 Validated NUMBER(1) CHECK (Validated IN (0,1)) NOT NULL,
 Lot NUMBER(1) CHECK (Lot IN (0,1)) NOT NULL,
 Valet NUMBER(1) CHECK (Valet IN (0,1)) NOT NULL,
 BusinessParking_id NUMBER(2) GENERATED ALWAYS as IDENTITY(START with 1 INCREMENT by 1),
 PRIMARY KEY (BusinessParking_id)
);

CREATE TABLE Has_BP
(
 Business_id CHAR(22) NOT NULL,
 BusinessParking_id NUMBER(2) NOT NULL,
 FOREIGN KEY (Business_id) REFERENCES Business(Business_id),
 FOREIGN KEY (BusinessParking_id) REFERENCES BusinessParking(BusinessParking_id),
 PRIMARY KEY (Business_id)
);

CREATE TABLE Music
(
 Dj NUMBER(1) CHECK (Dj IN (0,1)) NOT NULL,
 Background_music NUMBER(1) CHECK (Background_music IN (0,1)) NOT NULL,
 No_music NUMBER(1) CHECK (No_music IN (0,1)) NOT NULL,
 Jukebox NUMBER(1) CHECK (Jukebox IN (0,1)) NOT NULL,
 Live NUMBER(1) CHECK (Live IN (0,1)) NOT NULL,
 Video NUMBER(1) CHECK (Video IN (0,1)) NOT NULL,
 Karaoke NUMBER(1) CHECK (Karaoke IN (0,1)) NOT NULL,
 Music_id NUMBER(3) GENERATED ALWAYS as IDENTITY(START with 1 INCREMENT by 1),
 PRIMARY KEY (Music_id)
);

CREATE TABLE Has_M
(
 Business_id CHAR(22) NOT NULL,
 Music_id NUMBER(3) NOT NULL,
 FOREIGN KEY (Business_id) REFERENCES Business(Business_id),
 FOREIGN KEY (Music_id) REFERENCES Music(Music_id),
 PRIMARY KEY (Business_id)
```

```
);

CREATE TABLE Ambiance
(
 Romantic NUMBER(1) CHECK (Romantic IN (0,1)) NOT NULL,
 Intimate NUMBER(1) CHECK (Intimate IN (0,1)) NOT NULL,
 Classy NUMBER(1) CHECK (Classy IN (0,1)) NOT NULL,
 Hipster NUMBER(1) CHECK (Hipster IN (0,1)) NOT NULL,
 Divey NUMBER(1) CHECK (Divey IN (0,1)) NOT NULL,
 Touristy NUMBER(1) CHECK (Touristy IN (0,1)) NOT NULL,
 Trendy NUMBER(1) CHECK (Trendy IN (0,1)) NOT NULL,
 Upscale NUMBER(1) CHECK (Upscale IN (0,1)) NOT NULL,
 Casual NUMBER(1) CHECK (Casual IN (0,1)) NOT NULL,
 Ambiance_id NUMBER(3) GENERATED ALWAYS as IDENTITY(START with 1 INCREMENT by 1),
 PRIMARY KEY (Ambiance_id)
);

CREATE TABLE Has_A
(
 Business_id CHAR(22) NOT NULL,
 Ambiance_id NUMBER(3) NOT NULL,
 FOREIGN KEY (Business_id) REFERENCES Business(Business_id),
 FOREIGN KEY (Ambiance_id) REFERENCES Ambiance(Ambiance_id),
 PRIMARY KEY (Business_id)
);

CREATE TABLE DietaryRestrictions
(
 Dairy_free NUMBER(1) CHECK (Dairy_free IN (0,1)) NOT NULL,
 Gluten_free NUMBER(1) CHECK (Gluten_free IN (0,1)) NOT NULL,
 Vegan NUMBER(1) CHECK (Vegan IN (0,1)) NOT NULL,
 Kosher NUMBER(1) CHECK (Kosher IN (0,1)) NOT NULL,
 Halal NUMBER(1) CHECK (Halal IN (0,1)) NOT NULL,
 Soy_free NUMBER(1) CHECK (Soy_free IN (0,1)) NOT NULL,
 Vegetarian NUMBER(1) CHECK (Vegetarian IN (0,1)) NOT NULL,
 DietaryRestrictions_id NUMBER(3) GENERATED ALWAYS as IDENTITY(START with 1 INCREMENT by 1),
 PRIMARY KEY (DietaryRestrictions_id)
);

CREATE TABLE Has_DR
(
 Business_id CHAR(22) NOT NULL,
 DietaryRestrictions_id NUMBER(3) NOT NULL,
 FOREIGN KEY (Business_id) REFERENCES Business(Business_id),
 FOREIGN KEY (DietaryRestrictions_id) REFERENCES DietaryRestrictions(DietaryRestrictions_id),
 PRIMARY KEY (Business_id)
);
```

Cleaning the data: We link below the code we used to clean the data and the choices we made.

Processing Data_Business

April 26, 2020

In [2]:
```
import pandas as pd
import ast
import json
import re
import math
import unicodedata

business=pd.read_csv('yelp_academic_dataset_Business.csv')
print(business.dtypes)
```

```
address          object
attributes       object
business_id      object
categories       object
city             object
hours            object
is_open           int64
latitude        float64
longitude       float64
name             object
postal_code      object
review_count      int64
stars           float64
state            object
dtype: object
```

In [46]:
```
def satisfy_(array,condition):
    counter=0
    for idx,elem in enumerate(array):
        try:
            if not(condition(elem)):
                print("Not satisfied,",idx,"th element found was",elem)
                counter=counter+1
        except:
            print("Error occured,",idx,"th element found was",elem)
            counter=counter+1
```

```
            if counter==10:
                print("More than 10 Errors were found")
                return False

        return counter==0

    def satisfy(df,col,condition):
        return satisfy_(df[col],condition)
```

In [65]:
```
#To get better results in our database and reduce user input differences for same dat
#We set them all to lower case
#We delete the spaces at the end or beginning (strip)
#We delete all stress on letters (accents)
def parse_city(s):
    if s==s:
        return (''.join(c for c in unicodedata.normalize('NFD', s)
                  if unicodedata.category(c) != 'Mn')).lower().strip()
    else:
        return s

business['city'] = business['city'].apply(lambda x: parse_city(x))
```

In [50]:
```
# We check the addresses have 120 or less characters, cities less= 50, postal code le
# x!=x is used to check if the address is NaN (math.nan only works on ints), we accep
satisfy(business,'address',lambda x: x!=x or len(x)<=120)
satisfy(business,'city',lambda x: x!=x or len(x)<=50)
satisfy(business,'postal_code',lambda x: x!=x or len(x)<=12)

satisfy(business,'state',lambda x: len(x)<=3)
#We can see no business has empty state
```

Out[50]: True

In [51]:
```
# We check a business' name has 64 or less characters
satisfy(business,'name',lambda x: len(x)<=64)
#We can see no business has empty name
```

Out[51]: True

In [52]:
```
# We check there are no empty business_id, that they are 22 characters long and uniqu
satisfy(business, 'business_id', lambda x: len(x) == 22)
print(business['business_id'].is_unique)
```

```
True
```

In [6]:
```
# We check the star value in between 0 and 5, a review count is positive
satisfy(business, 'stars', lambda x: x >= 0 and x <= 5 and str(x)[1]=='.' and len(str(x
satisfy(business, 'review_count', lambda x: x >= 0 and x==int(x))
```

```
Out[6]: True

In [7]: # We check the is_open is only 0 or 1
        satisfy(business, 'is_open', lambda x: x == 0 or x==1)

Out[7]: True

In [8]: satisfy(business,'latitude',lambda x:len(str(x).split('.')[0]) in [1,2] and len(str(x)

Not satisfied, 1 th element found was 43.605498974300005
Not satisfied, 2 th element found was 51.041771000000004
Not satisfied, 4 th element found was 33.422191999999995
Not satisfied, 5 th element found was 35.1525551229
Not satisfied, 8 th element found was 33.420780199999996
Not satisfied, 9 th element found was 33.4928261717
Not satisfied, 10 th element found was 33.3776000741
Not satisfied, 11 th element found was 43.921109048999995
Not satisfied, 12 th element found was 36.108110499999995
Not satisfied, 15 th element found was 41.5029404472
More than 10 Errors were found


Out[8]: False
```

We need to round the latitude and longitude to 7 digits after the decimal point

```
In [9]: business['latitude']=business['latitude'].apply(lambda x: round(x*10000000)/10000000)
        business['longitude']=business['longitude'].apply(lambda x: round(x*10000000)/10000000)

In [10]: def inInterval(lower,upper,x):
             return lower<=x and x<=upper

In [11]: satisfy(business,'latitude',lambda x:inInterval(-90,90,int(str(x).split('.')[0])) and
         satisfy(business,'longitude',lambda x:inInterval(-180,180,int(str(x).split('.')[0])) a

Out[11]: True
```

Let's process the field Attributes

```
In [12]: #only keep necessary informations
         att=business[['business_id','attributes']].dropna(axis=0,how='any').set_index('busines
```

- We saw that some keys are assciated to 'None' values, we need to remove those keys because it does not give any information.

```
In [13]: def removeUseLessKeysAndNoiceLevel(string):
             final_dict={}
             dd=ast.literal_eval(string)
             for kk in dd.keys():
                 d=ast.literal_eval(dd[kk])
```

```
            if d!=None and type(d)!=str:
                final_dict[kk]=str(d)
        return str(final_dict)

    # Apply -> removeUseLessKeysAndNoiceLevel
    att['attributes']=att['attributes'].apply(lambda x: removeUseLessKeysAndNoiceLevel(x)
```

```
In [14]: def checkValues(key,dictio):
            subDict=ast.literal_eval(dictio[key])

            if key=='GoodForMeal':
                keysAreOk=satisfy_(subDict.keys(),lambda x:x in ['dessert','latenight', 'lunch
            elif key=='BusinessParking':
                keysAreOk=satisfy_(subDict.keys(),lambda x:x in ['garage','street', 'validated
            elif key=='Ambience':
                keysAreOk=satisfy_(subDict.keys(),lambda x:x in ['romantic','intimate', 'class
            elif key=='Music':
                keysAreOk=satisfy_(subDict.keys(),lambda x:x in ['dj','background_music', 'no_
            elif key=='DietaryRestrictions':
                keysAreOk=satisfy_(subDict.keys(),lambda x:x in ['dairy-free','gluten-free',
            else:#key Error
                return False

            valuesAreOK=satisfy_(subDict.keys(),lambda k:type(subDict[k])==bool)

            return keysAreOk and valuesAreOK
```

```
In [15]: satisfy(att,'attributes',lambda x:
                satisfy_(ast.literal_eval(x).keys(),lambda k:checkValues(k,ast.literal_eval(x)
```

```
Out[15]: True
```

```
In [16]: #remove empty dict
         att = att[att['attributes']!="{}"]
```

Let's create the auxiliary tables of the attributes

```
In [17]: att=business[['business_id','attributes']].dropna(axis=0,how='any')#.set_index('busine

         att['attributes']=att['attributes'].apply(lambda x: ast.literal_eval(x))
         #remove empty dict
         att = att[att['attributes']!={}]
```

```
In [18]: def removeKeysToNone(d):
            new_dict={}
            for k in d.keys():
                if d[k]!='None' and d[k]!='{}':
                    new_dict[k]=d[k]
            return new_dict
```

```
att['attributes']=att['attributes'].apply(lambda x:removeKeysToNone(x))
#remove empty dict
att = att[att['attributes']!={}]
```

In [19]:
```
def fromStringToAllInOne(string,columnsName):
    d=ast.literal_eval(string)
    allInOne=''
    for cn in columnsName:
        char='0'
        try:
            if d[cn]:
                char='1'
        except:
            pass
        allInOne=allInOne+char
    return allInOne

def createAttributeTable(attName,colName):
    #create the dataframe
    df = pd.DataFrame (columns = ['allInOne','id']+colName)
    #get all possible strings assciated with a key
    allPossibleString=[d[attName] for d in att['attributes'] if attName in d]
    #convert to format allInOne and remove duplicate
    df['allInOne']=list(dict.fromkeys([fromStringToAllInOne(string,colName) for string
    #generate the automatic increment that the database will produce when inserting ti
    df['id']=range(1,len(df['allInOne'])+1)
    #make the format allInOne as index in order to find it quicker having the allInOn
    df=df.set_index('allInOne')
    #populate the dataframe
    for aio in df.index:
        for idx,cn in enumerate(colName):
            df.loc[aio,cn]=int(aio[idx])

    return df

def createHas_Relation(attName,colName,attributeTable):
    #create dataframe
    Has_=pd.DataFrame (columns = ['business_id',attName+'_id'])
    #only keep those with the chosen key
    tempo=att[att['attributes'].apply(lambda d:attName in d.keys())]
    #extract business_ids in the final relation
    Has_['business_id']=tempo['business_id']
    #create a temporary Serie to perform operation
    tempo=tempo['attributes']
    #convert to format all_in_one
    tempo=tempo.apply(lambda d:fromStringToAllInOne(str(d[attName]),colName))
    #convert all_in_one into corresponding id
```

```
        tempo=tempo.apply(lambda aio:attributeTable.loc[aio]['id'])
        #set the attibute_tuple_id in the final relation
        Has_[attName+'_id']=tempo

        return Has_
```

In [20]:
```
#----------------------------GoodForMeal
attNameGFM='GoodForMeal'
colNameGFM=['dessert','latenight','lunch','dinner','brunch','breakfast']


GFM=createAttributeTable(attNameGFM,colNameGFM)
Has_GFM=createHas_Relation(attNameGFM,colNameGFM,GFM)

GFM=GFM.drop(columns=['id'])#because we don't need the fake index
#----------------------------BusinessParking
attNameBP='BusinessParking'
colNameBP=['garage','street', 'validated', 'lot', 'valet']


BP=createAttributeTable(attNameBP,colNameBP)
Has_BP=createHas_Relation(attNameBP,colNameBP,BP)

BP=BP.drop(columns=['id'])#because we don't need the fake index
#----------------------------Ambience
attNameA='Ambience'
colNameA=['romantic','intimate', 'classy', 'hipster', 'divey', 'touristy', 'trendy',


A=createAttributeTable(attNameA,colNameA)
Has_A=createHas_Relation(attNameA,colNameA,A)

A=A.drop(columns=['id'])#because we don't need the fake index
#----------------------------Music
attNameM='Music'
colNameM=['dj','background_music', 'no_music', 'jukebox', 'live', 'video', 'karaoke']


M=createAttributeTable(attNameM,colNameM)
Has_M=createHas_Relation(attNameM,colNameM,M)

M=M.drop(columns=['id'])#because we don't need the fake index
#----------------------------DietaryRestrictions
attNameDR='DietaryRestrictions'
colNameDR=['dairy-free','gluten-free', 'vegan', 'kosher', 'halal', 'soy-free', 'vegeta


DR=createAttributeTable(attNameDR,colNameDR)
```

```
Has_DR=createHas_Relation(attNameDR,colNameDR,DR)

DR=DR.drop(columns=['id'])#because we don't need the fake index
```

```
In [21]: #Import those tables
GFM.to_csv(r'GoodForMeal.csv', index = False)
Has_GFM.to_csv(r'Has_GFM.csv', index = False)
BP.to_csv(r'BusinessParking.csv', index = False)
Has_BP.to_csv(r'Has_BP.csv', index = False)
A.to_csv(r'Ambience.csv', index = False)
Has_A.to_csv(r'Has_A.csv', index = False)
M.to_csv(r'Music.csv', index = False)
Has_M.to_csv(r'Has_M.csv', index = False)
DR.to_csv(r'DietaryRestrictions.csv', index = False)
Has_DR.to_csv(r'Has_CR.csv', index = False)
```

Let's compute the NoiseLevel

```
In [22]: def fromDictToSanitizedNoiseLevel(dictio):
            if 'NoiseLevel' in dictio:
                s=dictio['NoiseLevel']
                if s[0]=='u':
                    s=s[1:]
                s=s[1:-1]
                return s
            return float('NaN')
```

```
In [23]: business['NoiseLevel']=att['attributes'].apply(lambda d:fromDictToSanitizedNoiseLevel
```

Let's build the tables with the opening hours

```
In [3]: hours = list(business['hours'])
list_of_timetable = list()
for i in hours:
    if i == i:
        a = ast.literal_eval(i)
        for b in a.keys():
            list_of_timetable.append((b,a[b].split("-",1)[0],a[b].split("-",1)[1]))
```

```
In [4]: set_of_timetable = set(list_of_timetable)
unique_list = list(set_of_timetable)
timetable = pd.DataFrame(unique_list, columns=['Day','Opening','Closing'])
```

```
In [30]: timetable.to_csv(r'Timetable.csv', index=False)
```

```
In [5]: list_business = business[['business_id' , 'hours']].values.tolist()
list_is_open = list()
for i in list_business:
    if i[1]==i[1]:
```

```
                b_id = i[0]
                a = ast.literal_eval(i[1])
                for b in a.keys():
                    list_is_open.append((unique_list.index((b,a[b].split("-",1)[0],a[b].split('
```

```
In [27]: is_open = pd.DataFrame(list_is_open, columns=['Timetable_Id','Business_Id'])
         is_open.to_csv(r'Is_open.csv', index=False)
```

Let's build the tables with cities, states and postal code

```
In [66]: city_table = business[['city', 'state']]

         print("In the beginning we have", city_table.count()[1], " data samples.")

         city_table = city_table[['city', 'state']].drop_duplicates()

         print("After dropping the duplicates we only have ", city_table.count()[1], " left.")

         city_table.head(10)
```

```
In the beginning we have 192609  data samples.
After dropping the duplicates we only have  1124  left.
```

Out[66]:

	city	state
0	phoenix	AZ
1	mississauga	ON
2	calgary	AB
3	las vegas	NV
4	tempe	AZ
5	charlotte	NC
6	montreal	QC
8	mesa	AZ
9	scottsdale	AZ
11	nobleton	ON

```
In [68]: #Creates the main City table
         city_table.to_csv(r'City.csv', index = False)
```

```
In [55]: postal_table = business[['city', 'state', 'postal_code']]
         print("In the beginning we have", postal_table.count()[2], "data samples.")

         postal_table = postal_table[['city', 'state', 'postal_code']].drop_duplicates()
         print("After dropping the duplicates we only have ", postal_table.count()[2], "left.")
```

```
In the beginning we have 191950 data samples.
After dropping the duplicates we only have  20041 left.
```

```
In [56]: list_postal = postal_table.values.tolist()
         city_list = city_table.values.tolist()
         list_postal_indexed = list()

         for i in list_postal:
             city_id = city_list.index([i[0],i[1]]) +1
             list_postal_indexed.append( (i[2] , city_id) )

In [57]: postal_table = pd.DataFrame(list_postal_indexed, columns=['Postal_code','City_id'])
         postal_table = postal_table[['Postal_code', 'City_id']].drop_duplicates()

         postal_table.head()

Out[57]:   Postal_code  City_id
         0       85016        1
         1     L5R 3E7        2
         2     T2R 1L3        3
         3       89149        4
         4       85281        5

In [59]: #Creates the Postal Code table
         postal_table.to_csv(r'Postal_code.csv', index=False, na_rep='NULL')

In [60]: list_business = business[['business_id', 'postal_code', 'city', 'state']].values.tolis
         postal_table = business[['city', 'state', 'postal_code']].drop_duplicates()
         full_list_postal = postal_table.values.tolist()
         list_is_in = list()

         for i in list_business:
             postal_id = full_list_postal.index([i[2],i[3],i[1]]) + 1
             list_is_in.append( (i[0], postal_id))

In [61]: #Creates the Is_in table
         Is_in_table = pd.DataFrame(list_is_in, columns=['Business_id','Postal_code_id'])
         Is_in_table.to_csv(r'Is_in.csv', index=False)
```

Let's process the final version of business

```
In [38]: business_table = business.drop(columns=['city','hours','postal_code','state','attribu
         business_table.head()

Out[38]:                                                address  \
         business_id
         1SWheh84yJXfytovILXOAQ   2818 E Camino Acequia Drive
         QXAEGFB4oINsVuTFxEYKFQ          30 Eglinton Avenue W
         fcXOEZdXYeZqnQ3lGlOXmg  1210 8th Street SW, Unit 220
         f2ZWZPENViL92BrFsIgR6w   6955 N Durango Dr, Ste 1116
         6KgGE8B1RsR7jc9R5nuHOQ             4 E University Dr
```

```
                                                                  categories  \
business_id
1SWheh84yJXfytovILXOAQ                                  Golf, Active Life
QXAEGFB4oINsVuTFxEYKFQ  Specialty Food, Restaurants, Dim Sum, Imported...
fcXOEZdXYeZqnQ3lGlOXmg  Local Services, Professional Services, Compute...
f2ZWZPENViL92BrFsIgR6w                        Beauty & Spas, Hair Salons
6KgGE8B1RsR7jc9R5nuHOQ              American (Traditional), Restaurants

                        is_open   latitude   longitude  \
business_id
1SWheh84yJXfytovILXOAQ        0  33.522143 -112.018481
QXAEGFB4oINsVuTFxEYKFQ        1  43.605499  -79.652289
fcXOEZdXYeZqnQ3lGlOXmg        1  51.041771 -114.081109
f2ZWZPENViL92BrFsIgR6w        1  36.287312 -115.289540
6KgGE8B1RsR7jc9R5nuHOQ        0  33.422192 -111.939615

                                           name  review_count  stars  \
business_id
1SWheh84yJXfytovILXOAQ   Arizona Biltmore Golf Club             5    3.0
QXAEGFB4oINsVuTFxEYKFQ   Emerald Chinese Restaurant           128    2.5
fcXOEZdXYeZqnQ3lGlOXmg  Nucleus Information Service             5    2.0
f2ZWZPENViL92BrFsIgR6w                 Amazing Cuts           116    4.5
6KgGE8B1RsR7jc9R5nuHOQ                 Ruby Tuesday             9    2.5

                       NoiseLevel
business_id
1SWheh84yJXfytovILXOAQ        NaN
QXAEGFB4oINsVuTFxEYKFQ       loud
fcXOEZdXYeZqnQ3lGlOXmg        NaN
f2ZWZPENViL92BrFsIgR6w        NaN
6KgGE8B1RsR7jc9R5nuHOQ        NaN
```

In [39]:
```
#Import the Business_table
business_table.to_csv(r'Business.csv', index=True,na_rep='NULL')
```

Processing Data_Review

April 26, 2020

In [1]:
```
import pandas as pd
import re

review=pd.read_csv('yelp_academic_dataset_review.csv')
print(review.dtypes)
review.head()
```

```
business_id     object
cool             int64
date            object
funny          float64
review_id       object
stars          float64
text            object
useful         float64
user_id         object
dtype: object
```

Out[1]:
```
               business_id  cool                 date  funny  \
0  WTqjgwHlXbSFevF32_DJVw     0  2016-11-09 20:09:03    0.0
1  b1b1eb3uo-w561D0ZfCEiQ     0  2018-01-30 23:07:38    0.0
2  3fw2X5bZYeW9xCz_zGh0Hg     5  2016-05-07 01:21:02    4.0
3  mRUVMJkUGxrByzMQ2Mu0pA     0  2017-12-15 23:27:08    1.0
4  dm6sO_Y8JdKTE1ZM955yug     0  2014-12-17 19:04:33    0.0

                review_id  stars  \
0  2TzJjDVDEuAW6MR5Vuc1ug    5.0
1  11a8sVPMUFtaC7_ABRkmtw    1.0
2  G7XHMxG0bx9oBJNECG4IFg    3.0
3  -I5umRTkhw15RqpKMl_o1Q    1.0
4  0AsmPiAQduxh5jE_si8cLA    5.0

                                                text  useful  \
0  I have to say that this office really has it t...     3.0
1  Today was my second out of three sessions I ha...     7.0
2  Tracy dessert had a big name in Hong Kong and ...     5.0
```

```
3  Walked in around 4 on a Friday afternoon, we s...     0.0
4  ended up here because Raku was closed and it r...     0.0

                  user_id
0  n6-Gk65cPZL6Uz8qRm3NYw
1  ssoyf2_x0EQMed6fgHeMyQ
2  jlu4CztcSxrKx56ba1a5AQ
3  -mA3-1mN4JIEkqOtdbNXCQ
4  C_hUvw2z0R-Rv0yZb6QCZA
```

```
In [2]: def satisfy(df,col,condition):
            counter=0
            for idx,elem in enumerate(df[col]):
                try:
                    if not(condition(elem)):
                        print("Not satisfied,",idx,"th element found was",elem)
                        counter=counter+1
                except:
                    print("Error occured,",idx,"th element found was",elem)
                    counter=counter+1
                if counter==20:
                    print("More than 20 Errors were found")
                    return

            print("Done")
```

```
In [3]: satisfy(review,'stars',lambda x:x>=0 and x<=5)

Not satisfied, 199242 th element found was nan
Done
```

```
In [4]: review.iloc[199242]

Out[4]: business_id                        NaN
        cool                                 1
        date           53bZ_EsXH71L7iFs5MP9_w
        funny                              NaN
        review_id                          NaN
        stars                              NaN
        text                               NaN
        useful                             NaN
        user_id                            NaN
        Name: 199242, dtype: object
```

Clearly this line is full of garbage, we will delete it

```
In [5]: review=review.drop(review.index[199242])
```

```
In [6]: satisfy(review,'stars',lambda x:x>=0 and x<=5)
```

```
Done
```

```
In [7]: satisfy(review,'user_id',lambda x:len(x)==22)
```

```
Error occured, 199241 th element found was nan
Done
```

```
In [8]: review.iloc[199241]
```

```
Out[8]: business_id                                  41b2SLmjLcxTGLVRxASiDA
        cool                                                              0
        date                                            2006-04-11 09:05:18
        funny                                                             0
        review_id                                    WWYQ1ce6mNt7AvRHu8w-jQ
        stars                                                             3
        text           Rating purely on food and 18th hole view: 5 stars
        useful                                                          NaN
        user_id                                                         NaN
        Name: 199241, dtype: object
```

In our standards we decided that the user id of a review could not be NULL, therefore we will remove this line

```
In [9]: review=review.drop(review.index[199241])
```

```
In [10]: satisfy(review,'user_id',lambda x:len(x)==22)
         satisfy(review,'review_id',lambda x:len(x)==22)
         satisfy(review,'business_id',lambda x:len(x)==22)
```

```
Done
Done
Done
```

```
In [11]: satisfy(review,'cool',lambda x:x==int(x))
         satisfy(review,'funny',lambda x:x==int(x))
         satisfy(review,'stars',lambda x:x==int(x))
         satisfy(review,'useful',lambda x:x==int(x))
```

```
Done
Done
Done
Done
```

```
In [12]: satisfy(review,'cool',lambda x:x>=0)
         satisfy(review,'funny',lambda x:x>=0)
         satisfy(review,'useful',lambda x:x>=0)
```

```
Done
Done
Done
```

```
In [13]: review=review.astype({'funny': 'int32','cool': 'int32','stars': 'int32','useful': 'int
         review.dtypes

Out[13]: business_id    object
         cool            int32
         date           object
         funny           int32
         review_id      object
         stars           int32
         text           object
         useful          int32
         user_id        object
         dtype: object
```

```
In [14]: #We check the dates have a correct format
         rex = re.compile("^[0-2]{1}[0-9]{3}[-]{1}[0-1]{1}[0-9]{1}[-]{1}[0-3]{1}[0-9]{1}[ ]{1}

         satisfy(review, 'date', lambda x: rex.match(x))

Done
```

```
In [15]: satisfy(review, 'text', lambda x: len(x)<=50)

Done
```

```
In [16]: review['review_id'].is_unique

Out[16]: True
```

```
In [17]: review=review.set_index('review_id')
```

```
In [18]: review.head()

Out[18]:                                    business_id  cool                 date  \
         review_id
         2TzJjDVDEuAW6MR5Vuc1ug  WTqjgwHlXbSFevF32_DJVw     0  2016-11-09 20:09:03
         11a8sVPMUFtaC7_ABRkmtw  b1b1eb3uo-w561D0ZfCEiQ     0  2018-01-30 23:07:38
         G7XHMxG0bx9oBJNECG4IFg  3fw2X5bZYeW9xCz_zGhOHg     5  2016-05-07 01:21:02
         -I5umRTkhw15RqpKMl_o1Q  mRUVMJkUGxrByzMQ2MuOpA     0  2017-12-15 23:27:08
         0AsmPiAQduxh5jE_si8cLA  dm6sO_Y8JdKTE1ZM955yug     0  2014-12-17 19:04:33

                                 funny  stars  \
         review_id
```

```
2TzJjDVDEuAW6MR5Vuc1ug      0      5
11a8sVPMUFtaC7_ABRkmtw      0      1
G7XHMxG0bx9oBJNECG4IFg      4      3
-I5umRTkhw15RqpKMl_o1Q      1      1
0AsmPiAQduxh5jE_si8cLA      0      5

                                                                  text  \
review_id
2TzJjDVDEuAW6MR5Vuc1ug  I have to say that this office really has it t...
11a8sVPMUFtaC7_ABRkmtw  Today was my second out of three sessions I ha...
G7XHMxG0bx9oBJNECG4IFg  Tracy dessert had a big name in Hong Kong and ...
-I5umRTkhw15RqpKMl_o1Q  Walked in around 4 on a Friday afternoon, we s...
0AsmPiAQduxh5jE_si8cLA  ended up here because Raku was closed and it r...

                        useful                 user_id
review_id
2TzJjDVDEuAW6MR5Vuc1ug       3  n6-Gk65cPZL6Uz8qRm3NYw
11a8sVPMUFtaC7_ABRkmtw       7  ssoyf2_x0EQMed6fgHeMyQ
G7XHMxG0bx9oBJNECG4IFg       5  jlu4CztcSxrKx56ba1a5AQ
-I5umRTkhw15RqpKMl_o1Q       0  -mA3-1mN4JIEkqOtdbNXCQ
0AsmPiAQduxh5jE_si8cLA       0  C_hUvw2z0R-Rv0yZb6QCZA
```

In [19]:
```
review.to_csv(r'SanitizedReview.csv', index = True)
```

Processing Data_User

April 26, 2020

In [1]:
```
import pandas as pd
import re

user=pd.read_csv('yelp_academic_dataset_User.csv')
print(user.dtypes)
user.head()
```

```
average_stars         float64
compliment_cool         int64
compliment_cute         int64
compliment_funny        int64
compliment_hot          int64
compliment_list         int64
compliment_more         int64
compliment_note         int64
compliment_photos       int64
compliment_plain        int64
compliment_profile      int64
compliment_writer       int64
cool                    int64
elite                  object
fans                    int64
friends                object
funny                   int64
name                   object
review_count            int64
useful                  int64
user_id                object
yelping_since          object
dtype: object
```

Out[1]:

	average_stars	compliment_cool	compliment_cute	compliment_funny	\
0	4.03	1	0	1	
1	3.63	1	0	1	
2	3.71	0	0	0	
3	4.85	0	0	0	

```
4           4.08                80                 0                 80

   compliment_hot  compliment_list  compliment_more  compliment_note  \
0               2                0                0                1
1               1                0                0                0
2               0                0                0                1
3               1                0                0                0
4              28                1                1               16

   compliment_photos  compliment_plain         ...           cool  \
0                  0                 1         ...             25
1                  0                 0         ...             16
2                  0                 0         ...             10
3                  0                 2         ...             14
4                  5                57         ...            665

                elite  fans  \
0      2015,2016,2017     5
1                 NaN     4
2                 NaN     0
3                 NaN     5
4  2015,2016,2017,2018    39

                                              friends  funny    name  \
0  ['wSByVbwME4MzgkJaFyfvNg', 'cpQmAgOWatghp14h1p...     17  Rashmi
1  ['xh9VekYUo5CgVBxySQ7OTw', 'L3Z-WvSvYXvTf-lzST...     22   Jenna
2  ['1IQ_d1RuMj8iIpcF2CDohA', 'lwhksSpgIyeYZor_Hl...      8   David
3  ['mXWEAK4ns5hdsD0b5EW_TQ', '_5kP7SOsLG2YNCJ9kM...      4  Angela
4  ['Kw4ZuLGWYHlocofX_HFL-g', 'yuEzhI3PX4CHfCnJsl...    279   Nancy

   review_count useful                 user_id        yelping_since
0            95     84  l6BmjZMeQD3rDxWUbiAiow  2013-10-08 23:11:33
1            33     48  4XChL029mKr5hydo79Ljxg  2013-02-21 22:29:06
2            16     28  bc8C_eETBWL0olvFSJJd0w  2013-10-04 00:16:10
3            17     30  dD0gZpBctWGdWo9WlGuhlA  2014-05-22 15:57:30
4           361   1114  MM4RJAeH6yuaN8oZDSt0RA  2013-10-23 07:02:50

[5 rows x 22 columns]
```

```
In [2]: def satisfy_(array,condition):
            counter=0
            for idx,elem in enumerate(array):
                try:
                    if not(condition(elem)):
                        print("Not satisfied,",idx,"th element found was",elem)
                        counter=counter+1
                except:
                    print("Error occured,",idx,"th element found was",elem)
```

```
                    counter=counter+1
                if counter==20:
                    print("More than 20 Errors were found")
                    return False

            return counter==0

        def satisfy(df,col,condition):
            return satisfy_(df[col],condition)
```

In [3]:
```
satisfy(user,'average_stars',lambda x:x>=0 and x<=5)
```

Out[3]: True

In [4]:
```
satisfy(user,'compliment_cool',lambda x:x>=0 and x==int(x))
satisfy(user,'compliment_cute',lambda x:x>=0 and x==int(x))
satisfy(user,'compliment_funny',lambda x:x>=0 and x==int(x))
satisfy(user,'compliment_hot',lambda x:x>=0 and x==int(x))
satisfy(user,'compliment_list',lambda x:x>=0 and x==int(x))
satisfy(user,'compliment_more',lambda x:x>=0 and x==int(x))
satisfy(user,'compliment_note',lambda x:x>=0 and x==int(x))
satisfy(user,'compliment_photos',lambda x:x>=0 and x==int(x))
satisfy(user,'compliment_plain',lambda x:x>=0 and x==int(x))
satisfy(user,'compliment_profile',lambda x:x>=0 and x==int(x))
satisfy(user,'compliment_writer',lambda x:x>=0 and x==int(x))
satisfy(user,'cool',lambda x:x>=0 and x==int(x))
satisfy(user,'fans',lambda x:x>=0 and x==int(x))
satisfy(user,'funny',lambda x:x>=0 and x==int(x))
satisfy(user,'review_count',lambda x:x>=0 and x==int(x))
satisfy(user,'useful',lambda x:x>=0 and x==int(x))
```

Out[4]: True

In [5]:
```
#We check the dates have a correct format
rex = re.compile("^[0-2]{1}[0-9]{3}[-]{1}[0-1]{1}[0-9]{1}[-]{1}[0-3]{1}[0-9]{1}[ ]{1}[0

satisfy(user,'yelping_since', lambda x: rex.match(x))
```

Out[5]: True

In [6]:
```
satisfy(user,'user_id',lambda x:len(x)==22)
```

Out[6]: True

In [7]:
```
#Check that all friends are ids of 22 characters and that friendship is not reflexive
satisfy(user,'friends',lambda x:satisfy_(x[2:-2].split("', '"),lambda y:len(y)==22 and
```

Out[7]: True

In [8]:
```
satisfy(user,'name',lambda x: len(x)<=45)
```

```
Error occured, 452522 th element found was nan
```

```
Out[8]: False
```

```
In [9]: user.iloc[452522]
```

```
Out[9]: average_stars                                    5
        compliment_cool                                  0
        compliment_cute                                  0
        compliment_funny                                 0
        compliment_hot                                   0
        compliment_list                                  0
        compliment_more                                  0
        compliment_note                                  0
        compliment_photos                                0
        compliment_plain                                 0
        compliment_profile                               0
        compliment_writer                                0
        cool                                             0
        elite                                          NaN
        fans                                             2
        friends                ['LrPLygQg5b5xZ8BwApOE8Q']
        funny                                            0
        name                                           NaN
        review_count                                     1
        useful                                           0
        user_id                       i0fRIsfCefhN_XdpgD9k4A
        yelping_since                    2016-05-20 21:24:18
        Name: 452522, dtype: object
```

This line is not a garbage line,this user has no name but we allow this in our database

```
In [10]: user['user_id'].is_unique
```

```
Out[10]: True
```

Let's build the Friend table

```
In [11]: #create a more practical dataframe
         friends=user.set_index('user_id')['friends']

         #compute the list of friends

         lst_user1=[u1 for u1 in user['user_id'] for f in friends[u1][2:-2].split("', '")if f>u
         lst_user2=[f for u1 in user['user_id'] for f in friends[u1][2:-2].split("', '") if f>u

         friends_final = pd.DataFrame (columns = ['user_id_1','user_id_2'])
         #populate the dataframe
```

```
friends_final['user_id_1']=lst_user1
friends_final['user_id_2']=lst_user2
friends_final.head()
```

Out[11]:

	user_id_1	user_id_2
0	l6BmjZMeQD3rDxWUbiAiow	wSByVbwME4MzgkJaFyfvNg
1	l6BmjZMeQD3rDxWUbiAiow	vRP9nQkYTeNioDjtxZlVhg
2	l6BmjZMeQD3rDxWUbiAiow	uWhC9eof98zPkvsalgaqJw
3	l6BmjZMeQD3rDxWUbiAiow	yW9QjWY0olv5-uRKv3t_Kw
4	4XChL029mKr5hydo79Ljxg	xh9VekYUo5CgVBxySQ70Tw

```
In [12]: #quite long, do not run it again by mistake
         friends_final.to_csv(r'Friend.csv', index = False)
```

Let's build the Is_elite table

```
In [13]: #create a more practical dataframe
         elite=user.set_index('user_id')['elite'].dropna()

         #compute the list of elite years
         lst_user_id_elite=[u1 for u1 in elite.index for f in elite[u1].split(",")]
         lst_elite=[f for u1 in elite.index for f in elite[u1].split(",")]

         elite_final = pd.DataFrame (columns = ['user_id','year'])
         #populate the dataframe
         elite_final['user_id']=lst_user_id_elite
         elite_final['year']=lst_elite

In [14]: #Do not run it again by mistake
         elite_final.to_csv(r'Is_elite.csv', index = False)
```

Let's continue to process the original table

```
In [15]: user=user.drop(columns=['elite','friends']).set_index('user_id')

In [16]: satisfy(user,'name',lambda x:x!='NULL')

Out[16]: True

In [17]: satisfy(user,'average_stars',lambda x:len(str(x)) in [3,4])

Not satisfied, 25 th element found was 3.5700000000000003
Not satisfied, 184 th element found was 4.8100000000000005
Not satisfied, 329 th element found was 3.5700000000000003
Not satisfied, 351 th element found was 3.5700000000000003
Not satisfied, 365 th element found was 3.5700000000000003
Not satisfied, 395 th element found was 3.5700000000000003
Not satisfied, 500 th element found was 4.5600000000000005
Not satisfied, 503 th element found was 4.5600000000000005
Not satisfied, 545 th element found was 3.5700000000000003
```

```
Not satisfied, 546 th element found was 4.8100000000000005
Not satisfied, 711 th element found was 3.5700000000000003
Not satisfied, 741 th element found was 4.5600000000000005
Not satisfied, 783 th element found was 2.5700000000000003
Not satisfied, 1038 th element found was 3.5700000000000003
Not satisfied, 1076 th element found was 4.8100000000000005
Not satisfied, 1086 th element found was 3.5700000000000003
Not satisfied, 1249 th element found was 4.8100000000000005
Not satisfied, 1283 th element found was 2.5700000000000003
Not satisfied, 1414 th element found was 4.5600000000000005
Not satisfied, 1431 th element found was 3.5700000000000003
More than 20 Errors were found
```

```
Out[17]: False
```

There seem to have some rounding issues

```
In [18]: user['average_stars']=user['average_stars'].apply(lambda x: round(x*100)/100)
```

```
In [23]: #check that it is rounded up to 1 or 2 number after the decimal point
         satisfy(user,'average_stars',lambda x:len(str(x).split('.')[0])==1 and len(str(x).spli
```

```
Out[23]: True
```

```
In [20]: #quite long, do not run it again by mistake
         user.to_csv(r'Users.csv', index = True)
```

Processing_Data_Tips

April 26, 2020

In [134]:
```
import pandas as pd
import re
```

In [135]:
```
tips=pd.read_csv('yelp_academic_dataset_tip.csv')
```

In [136]:
```
tips.set_index('business_id')

tips.head(5)
```

Out[136]:

	business_id	compliment_count	date	text	user_id
0	VaKXUpmWTTWDKbpJ3aQdMw	0	2014-03-27 03:51:24	Great for watching games, ufc, and whatever el...	UPw5DWs_b-e2JRBS-t37Ag
1	OPiPeoJiv92rENwbq76orA	0	2013-05-25 06:00:56	Happy Hour 2-4 daily with 1/2 price drinks and...	Ocha4kZBHb4JKOlOWvEOsg
2	5KheTjYPu1HcQzQFtm4_vw	0	2011-12-26 01:46:17	Good chips and salsa. Loud at times. Good serv...	jRyO2V1pA4CdVVqCIOPc1Q
3	TkoyGi8J7YFjA6SbaRzrxg	0	2014-03-23 21:32:49	The setting and decoration here is amazing. Co...	FuTJWFYm4UKqewaosss1KA
4	AkL6Ous6A1atZejfZXn1Bg	0	2012-10-06 00:19:27	Molly is definately taking a picture with Sant...	LUlKtaM3nXd-E4N4uOk_fQ

In [137]:
```
def satisfy_(array,condition):
    counter=0
    for idx,elem in enumerate(array):
        try:
            if not(condition(elem)):
                print("Not satisfied,",idx,"th element found was",elem)
                counter=counter+1
        except:
            print("Error occured,",idx,"th element found was",elem)
            counter=counter+1
        if counter==20:
            print("More than 20 Errors were found")
            return False

    return counter==0
```

In [138]:
```
def satisfy(df,col,condition):
    return satisfy_(df[col],condition)
```

In [139]:
```
# We check all compliment_counts are positive intergers
satisfy(tips, 'compliment_count', lambda x: x>=0)
satisfy(tips, 'compliment_count', lambda x: isinstance(x, int))
```

Out[139]: True

In [140]:
```
# We check there are no empty tips text (the x!=0 permits to prevent checking type f
satisfy(tips, 'text', lambda x: x!=0 and type(x)==str)
```

```
Not satisfied, 543883 th element found was nan
Not satisfied, 770499 th element found was nan
```

Out[140]: False

In [141]:
```
#A tip without text makes no sense so we can drop them
tips=tips.drop(tips.index[543883])
tips=tips.drop(tips.index[770499-1])
```

In [142]:
```
# Check we removed correctly
satisfy(tips, 'text', lambda x: x!=0 and type(x)==str)
```

Out[142]: True

In [143]:
```
def truncate(data_string):
    new_data = (data_string[:50]) if len(data_string) > 50 else data_string
    return new_data
```

In [144]:
```
# We need to truncate the text so they have at most 50 characters
tips['text'] = tips['text'].apply(lambda x: truncate(x))
```

In [145]:
```
# We check there are no more strings that have more than 50 characters
satisfy(tips, 'text', lambda x: len(x)<=50)
```

Out[145]: True

In [146]:
```
#We now further truncate the texts that have non UTF8 encoding
def truncateSpecialChar(s):
    while len(s.encode('utf-8'))>50:
        k=50/len(s.encode('utf-8'))
        s=s[:int(len(s)*k)]
    return s
```

In [147]:
```
tips['text']=tips['text'].apply(lambda x: truncateSpecialChar(x))
satisfy(tips, 'text', lambda x: len(x.encode('utf-8'))<=50)
```

Out[147]: True

In [148]:
```
#remove the new line character
tips['text']=tips['text'].apply(lambda x: x.replace('\n',''))
tips['text']=tips['text'].apply(lambda x: x.replace('\r',''))
satisfy(tips, 'text', lambda x: not("\n" in x))
satisfy(tips, 'text', lambda x: not("\r" in x))
```

Out[148]: True

In [149]:
```
satisfy(tips, 'text', lambda x: len(x.encode('utf-8'))<60)
```

Out[149]: True

In [150]:
```
#We check the dates have a correct format
rex = re.compile("^[0-2]{1}[0-9]{3}[-]{1}[0-1]{1}[0-9]{1}[-]{1}[0-3]{1}[0-9]{1}[ ]{1}

satisfy(tips, 'date', lambda x: len(x) != 0 )
satisfy(tips, 'date', lambda x: rex.match(x))
```

Out[150]: True

In [151]:
```
# We check there are no empty business_id or user_id and that they are 22 characters
satisfy(tips, 'user_id', lambda x: len(x) == 22)
satisfy(tips, 'business_id', lambda x: len(x) == 22)


tips['key_pairs'] = list(zip(tips.business_id, tips.user_id))
print(tips['key_pairs'].is_unique)

grouped_tips = tips.groupby('business_id')['user_id'].nunique()
sum = 0;
for i in range(0, len(grouped_tips)):
    sum += grouped_tips[i]

print("The key pairs (buisness_id, user_id) are not unique. There are about ", tips.

tips['key_triples'] = list(zip(tips.key_pairs, tips.date))
print(tips['key_triples'].is_unique)
print("If we now look at the triplet (buisness_id, user_id, date), we see they are u
```

```
False
The key pairs (buisness_id, user_id) are not unique. There are about  178740 duplicates, this a
We cannot use this pair as primary key.
True
If we now look at the triplet (buisness_id, user_id, date), we see they are unique and it makes
```

In [152]:
```
tips_table = tips.drop(columns=['key_triples','key_pairs']).set_index('date')
#quite long, do not run it again by mistake
tips_table.to_csv(r'Tips.csv', index = True)
```

In [ ]:

DIAS: Data-Intensive Applications and Systems Laboratory
School of Computer and Communication Sciences
Ecole Polytechnique Fédérale de Lausanne
Building BC, Station 14
CH-1015 Lausanne
URL: http://dias.epfl.ch/

Query Implementation

(x = query number, a. = description of logic, b. = SQL statement, c. = query result)

1.
   a. The query computes the average number of reviews for all users, to do this we take advantage of the Oracle operation AVG that computes the average.
   b. SELECT AVG (review_count)
      FROM users;
   c. Average = 37.6524

2.
   a. The query searches the number of businesses located in Alberta and Québec and counts them. To achieve this query, from Business we "go through" our Locations tables (Is_in, Postal_code, City) using intersections on the keys until we find our attribute of interest (the states).
   b. SELECT COUNT(*)
      FROM BUSINESS b, Is_in ii, Postal_code p, City c
      WHERE b.BUSINESS_ID=ii.business_id AND ii.postal_code_id=p.postal_code_id AND p.city_id=c.city_id AND (c.state='QC' OR c.state='AB');
   c. Nb_business = 17231

3.
   a. Finds the business' name with the maximum number of categories and displays both information. We find the number of categories for each business by counting the number of separators + 1, we can then select the maximum found and return its name.
      Note: we decided not to add a "Nb of categories" attribute for Business. We think it would add much complexity as the number would need to be updated automatically each time a business wants to change its categories. (also the number of categories seem to only lead to a small portion possible queries)
   b. SELECT b.name, regexp_count(b.categories, '[^,]+')
      FROM BUSINESS b
      WHERE regexp_count(b.categories, '[^,]+')=(SELECT MAX(regexp_count(b2.categories, '[^,]+')) from business b2);
   c. Name=Best Of The Best DJ's ; Nb_categories=37

4.
   a. Finds the number of businesses that have at least one of the two categories (Dry cleaners / Dry Cleaning). We can simply use the Oracle operation CONTAINS (that finds the number of a substring in a string) to check which business have these labels.
   b. CREATE INDEX business_categories ON business(categories) INDEXTYPE IS CTXSYS.CONTEXT;
      SELECT COUNT(*)
      FROM BUSINESS b

DIAS: Data-Intensive Applications and Systems Laboratory
School of Computer and Communication Sciences
Ecole Polytechnique Fédérale de Lausanne
Building BC, Station 14
CH-1015 Lausanne
URL: http://dias.epfl.ch/

```
WHERE CONTAINS(b.categories, 'Dry Cleaners', 1) > 0
OR CONTAINS(b.categories, 'Dry Cleaning', 2) > 0;
```

c. Nb_business=1448

5.

a. Finds the overall number of reviews for all the businesses that have more than 150 reviews and have at least 2 (any 2) dietary restriction categories. We start by only selecting the businesses that have more than 150 reviews then checking for each that it has two or more dietary restriction using their sum (as they are represented as "Boolean").

b.
```
SELECT SUM(b.review_count)
FROM BUSINESS b, DIETARYRESTRICTIONS d, has_dr h
WHERE b.review_count > 150 AND b.business_id=h.business_id AND
h.dietaryrestrictions_id=d.dietaryrestrictions_id
AND (d.dairy_free+d.gluten_free+d.halal+d.kosher+d.soy_free+d.vegan+d.vegetarian) >= 2;
```

c. Nb_reviews = 4663

6.

a. Finds users with most friends and displays both information. We simply count each user's number of friends, order then by descending order and display the top 10.

b.
```
SELECT * FROM(
  SELECT u.user_id, COUNT(*)
  FROM friend f, users u
  WHERE u.user_id=f.user_id_1 OR u.user_id=f.user_id_2
  GROUP BY u.user_id
  ORDER BY 2 DESC)
WHERE ROWNUM <= 10
```

c.

User_id =	nb_friends=
8DEyKVyplnOcSKx39vatbg	4919
Oi1qbcz2m2SnwUeztGYcnQ	4603
ZlOCmdFaMIF56FR-nWr_2A	4597
yLW8OrR8Ns4X1oXJmkKYgg	4437
YttDgOC9AlM4HcAlDsbB2A	4222
djxnI8Ux8ZYQJhiOQkrRhA	4211
qVc8ODYU5SZjKXVBgXdI7w	4134
iLjMdZi0Tm7DQxX1C1_2dg	4067
F_5_UNX-wrAFCXuAkBZRDw	3943
MeDuKsZcnI3IU2g7OlV-hQ	3923

7.

a. Finds the business name, number of stars, and the business review count of the top-5 businesses based on their review count that are currently open in the city of San Diego. Here we start by only looking at the opened businesses, then we go and find out their city so we

DIAS: Data-Intensive Applications and Systems Laboratory
School of Computer and Communication Sciences
Ecole Polytechnique Fédérale de Lausanne
Building BC, Station 14
CH-1015 Lausanne
URL: http://dias.epfl.ch/

drop all the ones not in San Diego. Finally we order those in descending order. In the end we only found one opened business in San Diego.

b.
```
SELECT * FROM(
  SELECT  b.name,b.stars,b.review_count
  FROM BUSINESS b, Is_in ii, postal_code pc, city c
  WHERE b.is_open=1 and b.business_id=ii.business_id and
ii.postal_code_id=pc.postal_code_id and pc.city_id=c.city_id and c.city='san diego'
  ORDER BY 3 DESC)
WHERE ROWNUM <= 5
```

c. Name=Brooks Photography ; nb_starts=1.5 ; review_count=35

8.

a. Finds the state with highest number of businesses. We find out each business' state so we can group them by state, sort them by descending order and return the state with most businesses.

b.
```
SELECT * FROM(
  SELECT c.state,COUNT(*)
  FROM BUSINESS b, Is_in ii, postal_code pc, city c
  WHERE b.business_id=ii.business_id and ii.postal_code_id=pc.postal_code_id and
pc.city_id=c.city_id
  GROUP BY c.state
  ORDER BY 2 DESC)
WHERE ROWNUM=1
```

c. State=AZ ; nb_business=56'686

9.

a. Find the total average of "average star" of elite users, grouped by the year in which they started to be elite users. To achieve this we

b.
```
SELECT  ie.year, ROUND(AVG( u.average_stars ),2) avg_stars
FROM users u,is_elite ie
WHERE u.user_id=ie.user_id AND ie.year=(
        Select MIN(ie2.year)
        From is_elite ie2
        WHERE ie2.user_id=u.user_id)
GROUP BY ie.year
```

c.

Year=	Average=
2009	3.74
2014	3.86
2011	3.78
2007	3.77
2010	3.76
2008	3.76

DIAS: Data-Intensive Applications and Systems Laboratory
School of Computer and Communication Sciences
Ecole Polytechnique Fédérale de Lausanne
Building BC, Station 14
CH-1015 Lausanne
URL: http://dias.epfl.ch/

2006	3.81
2012	3.79
2017	3.98
2015	3.89
2016	3.94
2013	3.83
2018	4.02

10.

a. Finds the top-10 opened businesses in New-York based on their median "star" rating. To achieve this we use Oracle's MEDIAN functionality over the stars and get only the businesses with true is_opened attribute and that have New York as city_id. We can now group and order them by descending order and display the top-10.
Again we only found one opened business in San Diego.

b.
```
SELECT name,medstars
FROM(
   SELECT b.name,b.business_id, MEDIAN(r.stars) medStars
   FROM review r, business b,is_in ii,postal_code pc, city c
   WHERE r.business_id=b.business_id and b.business_id=ii.business_id and
ii.postal_code_id=pc.postal_code_id and pc.city_id=c.city_id and c.city='new york' and
b.is_open=1
   GROUP BY  b.name,b.business_id
   ORDER BY 1 DESC)
```

c. Name = Safari Beauty Studio ; median_stars = 1

11.

a. Finds the minimum, maximum, mean, and median number of categories per business. To achieve this we can simply use Oracle's statistical methods MIN, MAX, AVG and MEDIAN over the number of categories of each business. (+see Note in query 3 concerning attribute number of categories).
Note2: here we did consider the businesses that have no categories at all (NULL values so would lead min to be 0).

b.
```
SELECT MIN(regexp_count(b.categories, '[^,]+')), MAX(regexp_count(b.categories,
'[^,]+')),AVG(regexp_count(b.categories, '[^,]+')), MEDIAN(regexp_count(b.categories, '[^,]+'))
FROM business b
```

c. Min=1 ; max=37 ; mean=4.10 ; median=4

12.

a. Finds the businesses in Las Vegas that have a 'valet' parking and are opened between '19:00' and '23:00' hours on Friday. We start by finding all businesses in Las Vegas (go through Locations tables), then for each we find the ones with Valet parking attribute (in the

DIAS: Data-Intensive Applications and Systems Laboratory
School of Computer and Communication Sciences
Ecole Polytechnique Fédérale de Lausanne
Building BC, Station 14
CH-1015 Lausanne
URL: http://dias.epfl.ch/

corresponding table) and finally, on the remaining we check the ones that are opened at the given hours-day.

b. SELECT b.name, b.stars, b.review_count
FROM Business b, timetable t, is_open io, is_in ii, postal_code pc, city c, has_bp h, businessparking bp
WHERE b.business_id=ii.business_id AND ii.postal_code_id=pc.postal_code_id AND pc.city_id=c.city_id AND c.city='las vegas'
AND b.business_id=h.business_id AND h.businessparking_id=bp.businessparking_id AND bp.valet=1
AND b.business_id=io.business_id AND io.timetable_id=t.timetable_id AND t.day='Friday' AND to_char(t.opening,'HH24:MI')<='19:00' AND to_char(t.closing,'HH24:MI')>='23:00';

c.

Name=	stars=	review_count=
Artsy Nannies	4.5	87
Firefly	4	119
Public School 702	4	1167
Waffle Bar	3.5	171
Bonnano's	4	141

General Comments

Again, we mainly worked in group over Discord or Zoom. We reviewed together the changes that were needed after Milestone 1 feedback. We split the different parts of the dataset to sanitize and create the new csv files that correspond to our tables in ER model.

DIAS: Data-Intensive Applications and Systems Laboratory
School of Computer and Communication Sciences
Ecole Polytechnique Fédérale de Lausanne
Building BC, Station 14
CH-1015 Lausanne
URL: http://dias.epfl.ch/

Deliverable 3

Assumptions

We assumed that the entered user inputs (like the city name) should not be too much changed.
For example, we have set the cities' names to lower case, dropped the stress (accents) and punctuation and stripped the extra spaces after the city's name since they were minor changes, it is a standardization of data. However, we think we should not further correct the given inputs since it would involve changing it based on some similarity measure that could involve false positives. It would also mean that we need to choose "the correct name" for each conflicting spelling.

Query Implementation

1.

   a. Count the number of business that are in Ontario (state 'ON'), have at least 6 reviews (we using review_count attribute as it gives the number of reviews for a business) and have a rating above 4.2 (we use stars attribute as it gives the aggregated number of stars for a business).

   b. SELECT COUNT(*)
   FROM Business b, Is_in ii, Postal_code pc, City c
   WHERE b.review_count>=6 AND b.stars>4.2 AND b.business_id=ii.business_id AND ii.postal_code_id=pc.postal_code_id AND pc.city_id=c.city_id AND c.state='ON'

   c. result: 2891
   runtime: 0.049s

2.

   a. What is the average difference in review scores for businesses that are considered "good for dinner" that have noise levels "loud" or "very loud", compared to ones with noise levels "average" or "quiet"?
   To perform this query, we calculated separately the average of the loud businesses and the quiet ones. For each we started by filtering the businesses that were not good for dinner (using IN keyword), then we only selected the businesses possessing the required noise level.
   We assumed a business' review score is his stars score.
   Also note that we could use a third "variable" (WITH gfd AS (…)) for the businesses that are good for dinner so they are only computed once (then use IN gfd for a1 and a2).

   b. WITH a1 AS
   (SELECT AVG(b.stars) avg1
   FROM Business b
   WHERE (b.noiselevel='loud' OR b.noiselevel='very_loud') AND b.business_id IN (

DIAS: Data-Intensive Applications and Systems Laboratory
School of Computer and Communication Sciences
Ecole Polytechnique Fédérale de Lausanne
Building BC, Station 14
CH-1015 Lausanne
URL: http://dias.epfl.ch/

```
        SELECT b.business_id
        FROM Business b, Goodformeal g, HAS_GFM hg
        WHERE b.business_id=hg.business_id AND hg.goodformeal_id=g.goodformeal_id AND
g.dinner=1 )),
a2 AS
  (SELECT AVG(b.stars) avg2
  FROM Business b
  WHERE (b.noiselevel='quiet' OR b.noiselevel='average') AND b.business_id IN (
        SELECT b.business_id
        FROM Business b, Goodformeal g, HAS_GFM hg
        WHERE b.business_id=hg.business_id AND hg.goodformeal_id=g.goodformeal_id AND
g.dinner=1 )
        )
SELECT avg1-avg2
FROM a1, a2
```

c. result: -0.3095317
runtime: 0.053s

3.

a. Find the business that contains the string "Irish Pub" in their "category" attribute, and keep the business that are mapped to a music that has its attribute "live"=1. We simply check all the businesses that contains the word Irish Pub in their categories (using CONTAINS) and they have live condition in Music.

b.
```
SELECT b.name,b.stars,b.review_count
FROM Business b,Has_M hm,Music m
WHERE CONTAINS(b.categories,'Irish Pub',1)>0 and b.business_id=hm.business_id and
hm.music_id=m.music_id and m.live=1
```

c. Result (26 overall, display 5 first):

Name	Stars	Review_Count
The Harp And Crown Pub And Kitchen	4	123
Grumpy's Bar	4	29
O'Connor's Pub	4	34
Hennessey's Tavern	3	384
Whelan's Gate Irish Pub	3	27

runtime: 0.009s

DIAS: Data-Intensive Applications and Systems Laboratory
School of Computer and Communication Sciences
Ecole Polytechnique Fédérale de Lausanne
Building BC, Station 14
CH-1015 Lausanne
URL: http://dias.epfl.ch/

4.

a. Find the average number of attribute “useful” of the users whose average rating falls in the following 2 ranges: [2-4), [4-5]. Display separately these results for elite users vs. regular users (4 values total).
We first compute the average number of attribute “useful” (using useful attribute) for the users that are elites and whose average rating (using average_stars attribute) falls in the range [2-4)
Then we compute the average number of attribute “useful” for the users that are not elites and whose average rating falls in the range [2-4)
Then we compute the average number of attribute “useful” for the users that are elites and whose average rating falls in the range [4-5]
Finally, we compute the average number of attribute “useful” for the users that are not elites and whose average rating falls in the range [4-5]

b.
```
WITH avg1 as (SELECT ROUND(AVG(u1.useful),5) a1
FROM Users u1
WHERE u1.user_id IN (SELECT ie1.user_id FROM Is_elite ie1) AND u1.average_stars<4 AND u1.average_stars>=2),

avg2 as (SELECT ROUND(AVG(u2.useful),5) a2
FROM Users u2
WHERE u2.user_id NOT IN (SELECT ie2.user_id FROM Is_elite ie2) AND u2.average_stars<4 AND u2.average_stars>=2),

avg3 as (SELECT ROUND(AVG(u3.useful),5) a3
FROM Users u3
WHERE u3.user_id IN (SELECT ie3.user_id FROM Is_elite ie3) AND u3.average_stars<=5 AND u3.average_stars>=4),

avg4 as (SELECT ROUND(AVG(u4.useful),5) a4
FROM Users u4
WHERE u4.user_id NOT IN (SELECT ie4.user_id FROM Is_elite ie4) AND u4.average_stars<=5 AND u4.average_stars>=4)
SELECT a1, a2, a3, a4
FROM avg1, avg2, avg3, avg4
```

c. result: 694.4353 34.89717 516.55523 14.13557

	elite	Not elite
Average ∈ [2,4)	694.4353	34.89717
Average ∈ [4,5]	516.55523	14.13557

runtime: 0,500s

DIAS: Data-Intensive Applications and Systems Laboratory
School of Computer and Communication Sciences
Ecole Polytechnique Fédérale de Lausanne
Building BC, Station 14
CH-1015 Lausanne
URL: http://dias.epfl.ch/

5.

a. Find the average rating and number of reviews for all businesses which have at least two categories and at least one parking type.
To achieve this, we start by filtering out all business that do not have at least one parking type (the sum of all Parking Type attribute should be 0 for those since each parking attribute is a Boolean (0/1)). On the remaining businesses we count their number of categories using regexp_count and check that they have at least two.

b.
```
SELECT AVG(b.review_count), AVG(b.stars)
FROM Business b, Has_BP hbp, BusinessParking bp
WHERE b.business_id = hbp.business_id AND hbp.businessParking_id=bp.businessParking_id AND
(bp.garage+bp.street+bp.validated+bp.lot+bp.valet) >= 1 AND
regexp_count(b.categories, '[^,]+') >= 2
```

c. result: AVG(review.count) = 72.871, AVG(stars) = 3.709
runtime: 1.614s

6.

a. What is the fraction of businesses that are considered "good for late night meals"?
To get the fraction we need to divide the number of business that are good for late night meal by the total number of business.
That is why the first query "count1" gives the number of business that are good for late night meal, whereas the
second query "count2" gives total number of business. We then divide the two subresults

b.
```
WITH count1 as(
SELECT COUNT(*) as nb
FROM  Business b,GoodForMeal gm,Has_GFM hgm
WHERE b.business_id=hgm.business_id and hgm.goodformeal_id=gm.goodformeal_id and gm.latenight=1
),
count2 as(
SELECT COUNT(*) as total
FROM Business b
)
SELECT nb/total
FROM count1,count2
```

DIAS: Data-Intensive Applications and Systems Laboratory
School of Computer and Communication Sciences
Ecole Polytechnique Fédérale de Lausanne
Building BC, Station 14
CH-1015 Lausanne
URL: http://dias.epfl.ch/

c. result: 0.01038892263
runtime: 0.014s

7.

a. Find the names of the cities where all businesses are closed on Sundays. To achieve this we simply select the cities where the number of business in this city that are open on Sunday is 0 (using our Timetable structure it is quite straight forward).

b.
```
SELECT c.city
FROM City c
WHERE (SELECT COUNT(*)
FROM Business b, postal_code pc, is_in ii, is_open io, timetable t
WHERE b.business_id=ii.business_id AND ii.postal_code_id=pc.postal_code_id AND pc.city_id=c.city_id
AND b.business_id=io.business_id AND io.timetable_id=t.timetable_id AND t.day='Sunday')=0
```

c. 277 results (5 first below):
summerlin south
las vegas valley
westworld scottsdale
suncity
montreal-quest
runtime: 0.133s

8.

a. Find the ids of the businesses that have been reviewed by more than 1030 unique users.
In order to do this we started by filtering out all business with less than 1030 review (it is an easy operation, it only needs to get the attribute review_count of business and it eliminates a good part of them). This is not necessary, however it accelerates the query. Then using Review we get all the users that voted for each of the remaining businesses. We only keep the distinct pairs. Finally we count and only select the ones that have more than 1030 reviewers.
We remarked that the review_count registered in Business were quite far from the actual number of Reviews corresponding.

DIAS: Data-Intensive Applications and Systems Laboratory
School of Computer and Communication Sciences
Ecole Polytechnique Fédérale de Lausanne
Building BC, Station 14
CH-1015 Lausanne
URL: http://dias.epfl.ch/

b.
```
SELECT un.business_id, un.c
FROM (
  SELECT u.business_id, COUNT(*) AS c
  FROM (
    SELECT DISTINCT b.business_id, u.user_id
    FROM Business b, Review r, Users u
    WHERE b.business_id = r.business_id AND r.user_id = u.user_id AND b.business_id IN (
      SELECT b.business_id
      FROM Business b
      WHERE b.review_count >= 1030
      )
  ) u
  GROUP BY u.business_id
) un
WHERE un.c >= 1030
```

c. results:
4JNXUYY8wbaaDmk3BPzlWw (with 1251 reviewers)
RESDUcs7fliihp38-d6_6g (with 1311 reviewers)
runtime: 0.146s

9.

a. Find the top-10 (by the number of stars) businesses in the state of California. We retrieve all business in the state of California and sort them in descending order by their number of stars and only keep the 10 first rows to get the top-10.

b.
```
SELECT b.name,b.stars
FROM Business b,Is_in ii,Postal_code pc,City c
WHERE b.business_id=ii.business_id and ii.postal_code_id=pc.postal_code_id and
pc.city_id=c.city_id and c.state='CA'
ORDER BY b.stars DESC
FETCH FIRST 10 ROWS ONLY
```

c. Results:

BusinessName	stars
Jaclyn Webb Healing	5
Beachside Tans	5
Finest-Edge Precision Sharpening Service	5
Fireplace Door Guy	5
Pretty Girl Lingo	5
Melody Events	5
Respclearance	5

DIAS: Data-Intensive Applications and Systems Laboratory
School of Computer and Communication Sciences
Ecole Polytechnique Fédérale de Lausanne
Building BC, Station 14
CH-1015 Lausanne
URL: http://dias.epfl.ch/

Goldeen Myofascial Release Therapy	5
Core Pest Solutions	4.5
Rebecca Vinacour Photography	4.5

time:0.038s

10.

a. Find the top-10 (by number of stars) ids of businesses per state. Show the results per state, in a descending order of number of stars.
We first order the business for each state by the number of stars (using stars attribute), then we select the first 10 business for each state.

b.
```
SELECT rs.business_id, rs.stars, rs.state
FROM
        (SELECT b.business_id, b.stars,c.state, Row_Number()OVER(PARTITION BY c.state
                ORDER BY b.stars DESC)AS rank
        FROM business b, Is_in ii, postal_code pc, city c
        WHERE b.business_id=ii.business_id AND ii.postal_code_id=pc.postal_code_id AND
        pc.city_id=c.city_id) rs
WHERE rank <= 10
```

c. 176 results: (here are the 5 first)
D7bFbnLY79FJNTr5uIHhnQ
D04cZA4yUmM0VTdmM6dN3w
jI5_RyahopRcot97bvBPjA
VtlcFOWMSUqp9LBgXosi_Q
40V_yimIPp5rI6vdKKhYSQ
runtime: 0.367s

11.

a. Find and display all the cities that satisfy the following: each business in the city has at least two reviews.
We start by selecting all unique cities (no need for distinct keyword here since they are already unique in the table). We then substract from this set all the cities that have at least one business with a review count inferior to 2.
With further investigation we can see that no business has less than 2 reviews (in review count). Also, we can see that many people entered the same city with spelling mistakes. See Assumptions.

DIAS: Data-Intensive Applications and Systems Laboratory
School of Computer and Communication Sciences
Ecole Polytechnique Fédérale de Lausanne
Building BC, Station 14
CH-1015 Lausanne
URL: http://dias.epfl.ch/

b.
```
(SELECT c2.CITY ,c2.STATE ,c2.CITY_ID
FROM City c2
)
MINUS
(SELECT DISTINCT c1.CITY ,c1.STATE ,c1.CITY_ID
FROM Business b, Postal_code p, City c1, Is_in ii
WHERE b.review_count < 2 AND b.business_id = ii.business_id AND ii.postal_code_id = p.postal_code_id AND p.city_id = c1.city_id
)
```

c. results: returns all cities, here are the 5 first rows

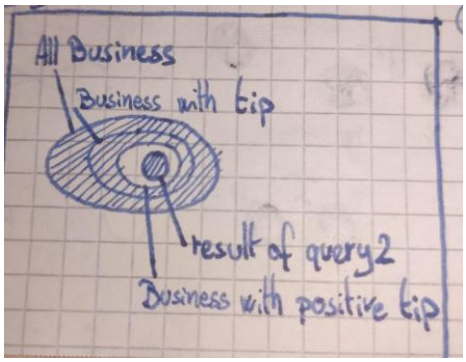
city	state	city_id
110 las vegas	NV	631
agincourt	ON	576
ahwahtukee	AZ	791
ahwatukee	AZ	488
ahwatukee foothills village	AZ	1054

runtime: 0.063s

12.

a. Find the number of businesses for which every user that gave the business a positive tip (containing 'awesome') has also given some business a positive tip within the previous day.
-The query count1 get the total number of business
-The query count2 get the number of business that have at least one positive tip associated
and that satisfy the complicated condition given in the instructions
-The query count3 get the number of business that have at least one positive tip
-details on the query count2:
Is made of two sub queries, one counts the number of positive tips by business (1)
and the other the number of positive tips that satisfy the condition that the same user made a
positive review without the previous 24hours (2)
then the global query counts the number of businesses for which the two numbers (1)(2) match
The result is computed as: (the total number of business)
- (business that have at least one positive tip)
+ (business that have at least one positive tip and that correspond to the condition)
<- This query is quite complicated because I noticed later that the query count2 alone would not include the business that has no positive review
Note: Note that businesses that do not have any tips containing "awesome" do satisfy the condition. In the Venn diagram the colored part is the part that satisfy the condition

DIAS: Data-Intensive Applications and Systems Laboratory
School of Computer and Communication Sciences
Ecole Polytechnique Fédérale de Lausanne
Building BC, Station 14
CH-1015 Lausanne
URL: http://dias.epfl.ch/



b.
```
WITH count1 as(
  SELECT COUNT(*) as total
  FROM Business b),
count2 as(
  SELECT COUNT(*) as count_condition
  FROM
    (SELECT COUNT(*) c,t1.business_id
    FROM TIP t1
    WHERE t1.text LIKE '%awesome%' and EXISTS(
      SELECT *
      FROM TIP t2
      WHERE t1.user_id=t2.user_id and t2.text LIKE '%awesome%' and t1.dated>t2.dated and
t2.dated>t1.dated-1)
    GROUP BY t1.business_id) nb_b1,
    (SELECT COUNT(*) c,t1.business_id
    FROM TIP t1
    WHERE t1.text LIKE '%awesome%'
    GROUP BY t1.business_id) nb_b2
  WHERE nb_b1.business_id=nb_b2.business_id and nb_b1.c=nb_b2.c),
count3 as(
        SELECT COUNT(*) as count_awesome
        FROM (
                SELECT t.business_id
                FROM TIP t
                WHERE t.text LIKE '%awesome%'
                GROUP BY t.business_id))
SELECT total-count_awesome+count_condition
FROM count1,count2,count3
```

c. result=183258
time=1.531s

DIAS: Data-Intensive Applications and Systems Laboratory
School of Computer and Communication Sciences
Ecole Polytechnique Fédérale de Lausanne
Building BC, Station 14
CH-1015 Lausanne
URL: http://dias.epfl.ch/

13.

a. Find the maximum number of different businesses any user has ever reviewed.
We first count for each user the number of different business he reviewed, then we select the maximum of what we did previously.

b.
```
SELECT MAX(c)
FROM
  (SELECT Count(a1.business_id) c, a1.user_id
  FROM
    (SELECT DISTINCT r.user_id, r.business_id
      FROM Review r) a1
  GROUP BY a1.user_id)
```

c. result: 793
runtime: 0.438s

14.

a. What is the difference between the average useful rating of reviews given by elite and non-elite users?
To find out the difference we start by separating the User table into two distinct tables: the users that have been elite and the non_elite users. We compute the average useful score on each of these tables and return the difference.
The difference seemed quite significative, but we computed separately the average of all users (77) and of the elite users (630) so the result seems to make sense.

b.
```
WITH avg_elite AS (
  SELECT AVG(u1.useful) AS avg_e
  FROM Users u1
  WHERE u1.user_id IN(SELECT e.user_id FROM IS_ELITE e)
  ),
avg_non_elite AS (
  SELECT AVG(u2.useful) AS avg_ne
  FROM Users u2
  WHERE u2.user_id NOT IN(SELECT e.user_id FROM IS_ELITE e)
  )
SELECT avg_e - avg_ne
FROM avg_elite, avg_non_elite
```

c. result: 608,11
runtime: 0.230s

DIAS: Data-Intensive Applications and Systems Laboratory
School of Computer and Communication Sciences
Ecole Polytechnique Fédérale de Lausanne
Building BC, Station 14
CH-1015 Lausanne
URL: http://dias.epfl.ch/

15.

a. List the name of the businesses that are currently 'open', possess a median star rating of 4.5 or above, considered good for 'brunch', and open on weekends.
We used a first subQuery that compute median stars for each business with a groupBy expression
Then we simply select the businesses that are good for brunch meal, that have a median star greater or equal to 4.5
and that belong to the set of business that are opened either on Saturday and Sunday
(This second subquery is computed as follow: we get all "timetable" that are open on Saturday or Sunday and then get the business_ids associated to those timetables)
Not that the second subquery yield duplicate which is not a problem as it is used like a set.
This is the optimal way of doing it because checking for duplicate with DISTINCT is costly, furthermore there cannot be more than 2 duplicates.

b.
```
SELECT b.name
FROM Business b,HAS_GFM hgfm,GOODFORMEAL gfm,
  (SELECT MEDIAN(r.stars) median_stars,r.business_id
  FROM Review r
  GROUP BY r.business_id) median_b
WHERE b.is_open=1 and b.business_id=median_b.business_id and
median_b.median_stars>=4.5 and b. business_id=hgfm.business_id and
hgfm.goodformeal_id=gfm.goodformeal_id and gfm.brunch=1 and b.business_id IN (
  SELECT io.business_id
  FROM IS_OPEN io,TIMETABLE t
  WHERE io.timetable_id=t.timetable_id and (t.day='Saturday' or t.day='Sunday'))
```

c. 921 results (5 first below):
Ladera Taverna y Cocina
Spirit Organic Cafe
Nekter Juice Bar
Rise & Shine - A Steak & Egg Place
Sabor Miami Cafe & Gallery

time=2.257s

16.

a. List the 'name', 'star' rating, and 'review_count' of the top-5 businesses in the city of 'los angeles' based on the average 'star' rating that serve both 'vegetarian' and 'vegan' food and open between '14:00' and '16:00' hours.
We first select the business that are in the city of 'los angeles',
serve both 'vegetarian' and 'vegan' food and are open between '14:00' and '16:00' hours.
We then compute their average star rating (using the stars attribute in review).

DIAS: Data-Intensive Applications and Systems Laboratory
School of Computer and Communication Sciences
Ecole Polytechnique Fédérale de Lausanne
Building BC, Station 14
CH-1015 Lausanne
URL: http://dias.epfl.ch/

Finally, we keep only the top 5 business and display the name, stars, and review count of those business.
We observed that they were no business in Los Angeles, 'vegetarian' and 'vegan' so it is "normal" that the result is empty.

b.
```
SELECT b2.name, b2.stars, b2.review_count
FROM Business b2
WHERE b2.business_id IN (
  SELECT 1 FROM (SELECT b.business_id, AVG(r.stars) avg1
  FROM Business b, Is_in ii, Postal_code pc, city c, Is_open io, timetable t, dietaryrestrictions d, has_dr hd, Review r
  WHERE b.business_id=ii.business_id AND ii.postal_code_id=pc.postal_code_id AND pc.city_id=c.city_id AND c.city='los angeles'
  AND b.business_id=io.business_id AND io.timetable_id=t.timetable_id AND to_char(t.opening,'HH24:MI')<='14:00' AND to_char(t.closing,'HH24:MI')>='16:00'
  AND b.business_id=hd.business_id AND hd.dietaryrestrictions_id=d.dietaryrestrictions_id AND d.vegan=1 AND d.vegetarian=1
  AND b.business_id=r.business_id
  GROUP BY b.business_id
  ORDER BY (avg1))
  WHERE ROWNUM <=5)
```

c. result: empty
runtime: 2.142s

17.

a. Compute the difference between the average 'star' ratings (using the Reviews in the corresponding table) of businesses considered 'good for dinner' with a (1) "divey" and (2) an "upscale" ambience.
To compute this we start by finding all businesses that are good for dinner, then from this set we create two new sets of businesses that are as well (1)"divey" and (2) "upscale".
We can now find the corresponding reviews for each business category and compute their corresponding average stars obtained from the database ratings.
We first nested the query but it was messier and the gain wasn't important.

b.
```
WITH gfd AS (
  SELECT b1.business_id
  FROM Business b1, GoodForMeal gfm, Has_GFM hgfm
  WHERE b1.business_id=hgfm.business_id AND hgfm.goodformeal_id = gfm.goodformeal_id AND gfm.dinner=1
  ),
```

DIAS: Data-Intensive Applications and Systems Laboratory
School of Computer and Communication Sciences
Ecole Polytechnique Fédérale de Lausanne
Building BC, Station 14
CH-1015 Lausanne
URL: http://dias.epfl.ch/

```
divey AS (
   SELECT b2.business_id
   FROM Business b2, gfd, Ambiance amb1, has_a ha1
   WHERE b2.business_id=ha1.business_id AND ha1.ambiance_id=amb1.ambiance_id AND
amb1.divey=1 AND b2.business_id IN gfd.business_id
   ),
upscale AS (
SELECT b3.business_id
FROM Business b3, gfd, Ambiance amb2, has_a ha2
WHERE b3.business_id=ha2.business_id AND ha2.ambiance_id=amb2.ambiance_id AND
amb2.upscale=1 AND b3.business_id IN gfd.business_id
),
avg_d AS (
   SELECT AVG(r.stars) AS avg_divey
   FROM divey, Review r
   WHERE divey.business_id = r.business_id
),
avg_u AS (
   SELECT AVG(r.stars) AS avg_upscale
   FROM upscale, Review r
   WHERE upscale.business_id = r.business_id
)
SELECT avg_divey - avg_upscale
FROM avg_d, avg_u
```

c. result: -0.28 (avg_divey = 3.80, avg_upscale = 4.08)
runtime: 0.241s

18.

a. Find the number of cities that satisfy the following: the city has at least five businesses and each of the top-5 (in terms of number of reviews) businesses in the city has a minimum of 100 reviews. We noticed that an important optimization could be done: we only considered businesses with 100 or more reviews. We then grouped by city with COUNT to get the number of business satisfying the condition by city, then we simply counted the number of cities with 5 or more businesses (that satisfied the condition)
<- Because: the top-5 businesses by number of reviews have more than a certain amount of reviews if and only if the number of businesses with more than a certain amount of reviews is greater or equal than 5

b.
```
SELECT COUNT(*)
FROM (SELECT COUNT(*) nb_business,c.city
```

DIAS: Data-Intensive Applications and Systems Laboratory
School of Computer and Communication Sciences
Ecole Polytechnique Fédérale de Lausanne
Building BC, Station 14
CH-1015 Lausanne
URL: http://dias.epfl.ch/

```
   FROM CITY c,POSTAL_CODE pc,IS_IN ii,Business b
   WHERE c.city IS NOT NULL and c.city_id=pc.city_id and pc.postal_code_id=ii.postal_code_id
and ii.business_id=b.business_id and b.review_count>99
   GROUP BY c.city) count1
WHERE count1.nb_business>5
```

c. result: 75
time=0.038

19.

a. Version 1 of the query (see index optimization for an optimized query):
Find the names of the cities that satisfy the following: the combined number of reviews for the top-100 (by reviews) businesses in the city is at least double the combined number of reviews for the rest of the businesses in the city.
There is 2 main subquery the two are similar except that the first only consider the top-100 businesses by city in its sum whereas the second only consider the out-of-top-100 businesses by city in its sum. Note that we used the NVL function to consider as zero a sum of an empty set of values.
Each subquery yield the list of the review_count for each business in a given city with a rownumber corresponding to the sorted indexing by review_count, which will enable us to consider only the first 100 rows of those after that.
After very end we output the city names for which the first query sum is as least twice the second query sum

b.
```
SELECT  c.city
FROM City c
WHERE (SELECT SUM(r1.review_count)
      FROM(SELECT b1.review_count,ROW_NUMBER() OVER(ORDER BY b1.review_count DESC)
as rn
         FROM Business b1, Is_in ii1, Postal_code pc1
         WHERE b1.business_id=ii1.business_id AND ii1.postal_code_id=pc1.postal_code_id AND
pc1.city_id=c.city_id) r1
      WHERE r1.rn <= 100) >= 2*
      NVL((SELECT SUM(r2.review_count)
         FROM(SELECT b.review_count ,ROW_NUMBER() OVER(ORDER BY b.review_count DESC)
as rn
         FROM Business b, Is_in ii, Postal_code pc
         WHERE b.business_id=ii.business_id AND ii.postal_code_id=pc.postal_code_id AND
pc.city_id=c.city_id) r2
      WHERE r2.rn > 100),0)
```

DIAS: Data-Intensive Applications and Systems Laboratory
School of Computer and Communication Sciences
Ecole Polytechnique Fédérale de Lausanne
Building BC, Station 14
CH-1015 Lausanne
URL: http://dias.epfl.ch/

c. 1087results: 5 first:
mantua
phx
enterprise
west view
morin-heights
runtime: 31.06

20.

a. For each of the top-10 (by the number of reviews) businesses, find the top-3 reviewers by activity among those who reviewed the business. Reviewers by activity are defined and ordered as the users that have the highest numbers of total reviews across all the businesses (the users that review the most).
The subquery b10 takes the top-10 business ordered by review_count,
Then we joined the top10 business with their corresponding user (that reviewed the given business) and add a rownumber obtained by partitioning by business and ordered the users by their review_count
At the end, keeping the tuple with ranking smaller than 4 yield the top3 users by businesses considered as wanted

b.
```
SELECT data.business_id, data.user_id
FROM
  (SELECT b10.business_id,u.user_id,ROW_NUMBER() OVER(PARTITION BY b10.business_id
ORDER BY u.review_count DESC) AS row_num
  FROM
    (SELECT b.review_count,b.business_id
    FROM Business b
    ORDER BY 1 DESC
    FETCH FIRST 10 ROWS ONLY) b10,
    Review r,Users u
  WHERE b10.business_id=r.business_id and r.user_id=u.user_id) data
WHERE row_num<4
```

c. results:

Business_id	User_id rank
2weQS-RnoOBhb1KsHKyoSQ	bLbSNkLggFnqwNNzzq-Ijw 1
2weQS-RnoOBhb1KsHKyoSQ	nzsv-p1O8gCfP3XijfQrIw 2
2weQS-RnoOBhb1KsHKyoSQ	whINg-cC-FiAv_ATDGMDTg 3
4JNXUYY8wbaaDmk3BPzlWw	cMEtAiW60I5wE_vLfTxoJQ 1
4JNXUYY8wbaaDmk3BPzlWw	YttDgOC9AlM4HcAlDsbB2A 2
4JNXUYY8wbaaDmk3BPzlWw	ZlOCmdFaMIF56FR-nWr_2A 3
5LNZ67Yw9RD6nf4_UhXOjw	IucvvxdQXXhjQ4z6Or6Nrw 1

DIAS: Data-Intensive Applications and Systems Laboratory
School of Computer and Communication Sciences
Ecole Polytechnique Fédérale de Lausanne
Building BC, Station 14
CH-1015 Lausanne
URL: http://dias.epfl.ch/

5LNZ67Yw9RD6nf4_UhXOjw	CQUDh80m48xnzUkx-X5NAw 2
5LNZ67Yw9RD6nf4_UhXOjw	whINg-cC-FiAv_ATDGMDTg 3
DkYS3arLOhA8si5uUEmHOw	hWDybu_KvYLSdEFzGrniTw 1
DkYS3arLOhA8si5uUEmHOw	Xwnf20FKuikiHcSpcEbpKQ 2
DkYS3arLOhA8si5uUEmHOw	m07sy7eLtOjVdZ8oN9JKag 3
K7lWdNUhCbcnEvI0NhGewg	AbMjnKOwg736fcIu8apuyQ 1
K7lWdNUhCbcnEvI0NhGewg	s3kRi7b8t2sdtYcsMbqlJA 2
K7lWdNUhCbcnEvI0NhGewg	9akppeqi5dnaIqPyJ75aCw 3
RESDUcs7fIiihp38-d6_6g	RtGqdDBvvBCjcu5dUqwfzA 1
RESDUcs7fIiihp38-d6_6g	bLbSNkLggFnqwNNzzq-Ijw 2
RESDUcs7fIiihp38-d6_6g	NeXRy1C7PxS7bvIcQ3SymA 3
cYwJA2A6I12KNkm2rtXd5g	Xj0O2l0bp633ebmG468aZw 1
cYwJA2A6I12KNkm2rtXd5g	bLbSNkLggFnqwNNzzq-Ijw 2
cYwJA2A6I12KNkm2rtXd5g	iCYMf_sHRevmzWg8la-LVw 3
f4x1YBxkLrZg652xt2KR5g	0QeJC2inz6P-OVzROU_LNw 1
f4x1YBxkLrZg652xt2KR5g	IhmBr-yHC8xLjdnoq7O37w 2
f4x1YBxkLrZg652xt2KR5g	EDBTE8HMO-iN5hAi9B382A 3
iCQpiavjjPzJ5_3gPD5Ebg	bQCHF5rn5lMI9c5kEwCaNA 1
iCQpiavjjPzJ5_3gPD5Ebg	PLOYtrCMUFPHQe2IbYAd5g 2
iCQpiavjjPzJ5_3gPD5Ebg	9akppeqi5dnaIqPyJ75aCw 3
ujHiaprwCQ5ewziu0Vi9rw	W7DHyQlY_kXls2iXt-_2Ag 1
ujHiaprwCQ5ewziu0Vi9rw	NeXRy1C7PxS7bvIcQ3SymA 2
ujHiaprwCQ5ewziu0Vi9rw	AbMjnKOwg736fcIu8apuyQ 3

time = 0.186

DIAS: Data-Intensive Applications and Systems Laboratory
School of Computer and Communication Sciences
Ecole Polytechnique Fédérale de Lausanne
Building BC, Station 14
CH-1015 Lausanne
URL: http://dias.epfl.ch/

Query Performance Analysis – Indexing

Query 10:

Initial runtime: 0.367s

Optimized running time: 0.276s

Explain the improvement:
The I/O cost before the creation of the index
5326+5326+5326+2101+3+2098+17+2080+1128+239=23644
The I/O cost after the creation of the index
4472+4472+4472+1248+3+1244+17+1227+275+239=17669
The ratio
17669/23644=0.747%
Because now it is easier to retrieve business ordered by their stars, we are able to perform an index fast full scan instead of doing a fullscan of the business table.
If we look at the Operation 8, it takes 1128 I/O in the first version whereas the second version only need 275 I/O.

```
-------------------------------------------------------------------------------------------------
| Id  | Operation                | Name        | Rows  | Bytes |TempSpc| Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT         |             |   191K|  6916K|       |  5326   (1)| 00:00:01 |
|*  1 |  VIEW                    |             |   191K|  6916K|       |  5326   (1)| 00:00:01 |
|*  2 |   WINDOW SORT PUSHED RANK|             |   191K|    12M|    14M|  5326   (1)| 00:00:01 |
|*  3 |    HASH JOIN             |             |   191K|    12M|       |  2101   (1)| 00:00:01 |
|   4 |     TABLE ACCESS FULL    | CITY        |  1124 |  8992 |       |     3   (0)| 00:00:01 |
|*  5 |     HASH JOIN            |             |   191K|    11M|       |  2098   (1)| 00:00:01 |
|   6 |      TABLE ACCESS FULL   | POSTAL_CODE | 20117 |   176K|       |    17   (0)| 00:00:01 |
|*  7 |      HASH JOIN           |             |   192K|     9M|  7336K|  2080   (1)| 00:00:01 |
|   8 |       TABLE ACCESS FULL  | BUSINESS    |   192K|  5078K|       |  1128   (1)| 00:00:01 |
|   9 |       TABLE ACCESS FULL  | IS_IN       |   192K|  5078K|       |   239   (1)| 00:00:01 |
-------------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   1 - filter("RANK"<=10)
   2 - filter(ROW_NUMBER() OVER ( PARTITION BY "C"."STATE" ORDER BY
              INTERNAL_FUNCTION("B"."STARS") DESC )<=10)
   3 - access("PC"."CITY_ID"="C"."CITY_ID")
   5 - access("II"."POSTAL_CODE_ID"="PC"."POSTAL_CODE_ID")
   7 - access("B"."BUSINESS_ID"="II"."BUSINESS_ID")

Note
-----
   - this is an adaptive plan
```

DIAS: Data-Intensive Applications and Systems Laboratory
School of Computer and Communication Sciences
Ecole Polytechnique Fédérale de Lausanne
Building BC, Station 14
CH-1015 Lausanne
URL: http://dias.epfl.ch/

```
CREATE INDEX Business_stars_id
ON business(stars,business_id)


Plan hash value: 2364551333

-------------------------------------------------------------------------------------------------------------
| Id  | Operation                  | Name              | Rows  | Bytes |TempSpc| Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT           |                   |   191K|  6916K|       |  4472   (1)| 00:00:01 |
|*  1 |  VIEW                      |                   |   191K|  6916K|       |  4472   (1)| 00:00:01 |
|*  2 |   WINDOW SORT PUSHED RANK  |                   |   191K|    12M|    14M|  4472   (1)| 00:00:01 |
|*  3 |    HASH JOIN               |                   |   191K|    12M|       |  1248   (1)| 00:00:01 |
|   4 |     TABLE ACCESS FULL      | CITY              |  1124 |  8992 |       |     3   (0)| 00:00:01 |
|*  5 |     HASH JOIN              |                   |   191K|    11M|       |  1244   (1)| 00:00:01 |
|   6 |      TABLE ACCESS FULL     | POSTAL_CODE       | 20117 |   176K|       |    17   (0)| 00:00:01 |
|*  7 |      HASH JOIN             |                   |   192K|     9M|  7336K|  1227   (1)| 00:00:01 |
|   8 |       INDEX FAST FULL SCAN | BUSINESS_STARS_ID |   192K|  5078K|       |   275   (1)| 00:00:01 |
|   9 |       TABLE ACCESS FULL    | IS_IN             |   192K|  5078K|       |   239   (1)| 00:00:01 |
-------------------------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   1 - filter("RANK"<=10)
   2 - filter(ROW_NUMBER() OVER ( PARTITION BY "C"."STATE" ORDER BY
              INTERNAL_FUNCTION("B"."STARS") DESC )<=10)
   3 - access("PC"."CITY_ID"="C"."CITY_ID")
   5 - access("II"."POSTAL_CODE_ID"="PC"."POSTAL_CODE_ID")
   7 - access("B"."BUSINESS_ID"="II"."BUSINESS_ID")

Note
-----
   - this is an adaptive plan
```

DIAS: Data-Intensive Applications and Systems Laboratory
School of Computer and Communication Sciences
Ecole Polytechnique Fédérale de Lausanne
Building BC, Station 14
CH-1015 Lausanne
URL: http://dias.epfl.ch/

Query 14:
Initial runtime: 0.957s

Optimized running time: 0.668s

Explain the improvement: By creating this index, the cost is reduced by 18%. This is because in the new plan, we don't have to access the full 'users' table instead we can use this new index to speed up the process.

Initial plan:

Id	Operation	Name	Rows	Bytes	TempSpc	Cost (%CPU)	Time
0	SELECT STATEMENT		1	26		17513 (1)	00:00:01
1	TEMP TABLE TRANSFORMATION						
2	LOAD AS SELECT (CURSOR DURATION MEMORY)	SYS_TEMP_0FD9D6A4A_A7A78D5					
3	HASH UNIQUE		70704	1588K	7016K	1125 (1)	00:00:01
4	TABLE ACCESS FULL	IS_ELITE	223K	5012K		273 (1)	00:00:01
5	NESTED LOOPS		1	26		16388 (1)	00:00:01
6	VIEW		1	13		4058 (1)	00:00:01
7	SORT AGGREGATE		1	50			
* 8	HASH JOIN		70704	3452K	2488K	4058 (1)	00:00:01
9	VIEW		70704	1657K		62 (0)	00:00:01
10	TABLE ACCESS FULL	SYS_TEMP_0FD9D6A4A_A7A78D5	70704	1588K		62 (0)	00:00:01
11	TABLE ACCESS FULL	USERS	778K	19M		2472 (1)	00:00:01
12	VIEW		1	13		12330 (1)	00:00:01
13	SORT AGGREGATE		1	50			
* 14	HASH JOIN		778K	37M	26M	12330 (1)	00:00:01
15	VIEW		778K	17M		7127 (1)	00:00:01
16	MINUS						
17	SORT UNIQUE		778K	17M	23M		
18	INDEX FAST FULL SCAN	SYS_C00113620	778K	17M		1252 (1)	00:00:01
19	SORT UNIQUE		70704	1657K	2232K		
20	VIEW		70704	1657K		62 (0)	00:00:01
21	TABLE ACCESS FULL	SYS_TEMP_0FD9D6A4A_A7A78D5	70704	1588K		62 (0)	00:00:01
22	TABLE ACCESS FULL	USERS	778K	19M		2472 (1)	00:00:01

```
Predicate Information (identified by operation id):
---------------------------------------------------

   8 - access("U1"."USER_ID"="ELITE"."USER_ID")
  14 - access("U2"."USER_ID"="NON_ELITE"."USER_ID")
```

DIAS: Data-Intensive Applications and Systems Laboratory
School of Computer and Communication Sciences
Ecole Polytechnique Fédérale de Lausanne
Building BC, Station 14
CH-1015 Lausanne
URL: http://dias.epfl.ch/

```
-------------------------------------------------------------------------------------------------------------------------
| Id  | Operation                                | Name                       | Rows  | Bytes |TempSpc| Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                         |                            |     1 |    26 |       | 14594   (1)| 00:00:01 |
|   1 |  TEMP TABLE TRANSFORMATION               |                            |       |       |       |            |          |
|   2 |   LOAD AS SELECT (CURSOR DURATION MEMORY)| SYS_TEMP_0FD9D6605_C021605 |       |       |       |            |          |
|   3 |    HASH UNIQUE                           |                            | 70704 |  1588K|  7016K|  1125   (1)| 00:00:01 |
|   4 |     TABLE ACCESS FULL                    | IS_ELITE                   |  223K|  5012K|       |   273   (1)| 00:00:01 |
|   5 |   NESTED LOOPS                           |                            |     1 |    26 |       | 13469   (1)| 00:00:01 |
|   6 |    VIEW                                  |                            |     1 |    13 |       |  2678   (1)| 00:00:01 |
|   7 |     SORT AGGREGATE                       |                            |     1 |    50 |       |            |          |
|*  8 |      HASH JOIN                           |                            | 70704 |  3452K|  2488K|  2678   (1)| 00:00:01 |
|   9 |       VIEW                               |                            | 70704 |  1657K|       |    62   (0)| 00:00:01 |
|  10 |        TABLE ACCESS FULL                 | SYS_TEMP_0FD9D6605_C021605 | 70704 |  1588K|       |    62   (0)| 00:00:01 |
|  11 |       INDEX FAST FULL SCAN               | USER_ID_USEFUL             |  778K|    19M|       |  1092   (1)| 00:00:01 |
|  12 |    VIEW                                  |                            |     1 |    13 |       | 10791   (1)| 00:00:01 |
|  13 |     SORT AGGREGATE                       |                            |     1 |    50 |       |            |          |
|* 14 |      HASH JOIN                           |                            |  778K|    37M|    26M| 10791   (1)| 00:00:01 |
|  15 |       VIEW                               |                            |  778K|    17M|       |  6967   (1)| 00:00:01 |
|  16 |        MINUS                             |                            |       |       |       |            |          |
|  17 |         SORT UNIQUE                      |                            |  778K|    17M|    23M|            |          |
|  18 |          INDEX FAST FULL SCAN            | USER_ID_USEFUL             |  778K|    17M|       |  1092   (1)| 00:00:01 |
|  19 |         SORT UNIQUE                      |                            | 70704 |  1657K|  2232K|            |          |
|  20 |          VIEW                            |                            | 70704 |  1657K|       |    62   (0)| 00:00:01 |
|  21 |           TABLE ACCESS FULL              | SYS_TEMP_0FD9D6605_C021605 | 70704 |  1588K|       |    62   (0)| 00:00:01 |
|  22 |       INDEX FAST FULL SCAN               | USER_ID_USEFUL             |  778K|    19M|       |  1092   (1)| 00:00:01 |
-------------------------------------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   8 - access("U1"."USER_ID"="ELITE"."USER_ID")
  14 - access("U2"."USER_ID"="NON_ELITE"."USER_ID")
```

DIAS: Data-Intensive Applications and Systems Laboratory
School of Computer and Communication Sciences
Ecole Polytechnique Fédérale de Lausanne
Building BC, Station 14
CH-1015 Lausanne
URL: http://dias.epfl.ch/

Query 15:
Initial runtime: 2.257s

Optimized running time: 1.037s

Explain the improvement: We can see that we gain a significant cost / IOs. It comes from the fact that the database now recuperates the stars associated with each business very easily using the index: it can now read sequentially the stars of the reviews for each business (review for one given business are contiguous).

Initial plan:

```
Plan hash value: 2218229067
-------------------------------------------------------------------------------------------
| Id  | Operation             | Name       | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT      |            |   539 | 67914 |  8433   (1)| 00:00:01 |
|*  1 |  HASH JOIN SEMI       |            |   539 | 67914 |  8433   (1)| 00:00:01 |
|*  2 |   HASH JOIN           |            |   586 | 59772 |  7284   (1)| 00:00:01 |
|*  3 |    TABLE ACCESS FULL  | GOODFORMEAL |    30 |   180 |     3   (0)| 00:00:01 |
|*  4 |    HASH JOIN          |            |  1231 |   115K|  7281   (1)| 00:00:01 |
|*  5 |     HASH JOIN         |            |  6565 |   448K|  7213   (1)| 00:00:01 |
|   6 |      VIEW             |            |  6565 |   153K|  6084   (1)| 00:00:01 |
|*  7 |       FILTER          |            |       |       |            |          |
|   8 |        SORT GROUP BY  |            |  6565 |   166K|  6084   (1)| 00:00:01 |
|   9 |         TABLE ACCESS FULL| REVIEW  |   918K|    22M|  6062   (1)| 00:00:01 |
|* 10 |      TABLE ACCESS FULL | BUSINESS  |   158K|  7121K|  1128   (1)| 00:00:01 |
|  11 |     TABLE ACCESS FULL  | HAS_GFM   | 29714 |   754K|    68   (0)| 00:00:01 |
|  12 |   VIEW                 | VW_NSO_1  |   260K|  6112K|  1148   (1)| 00:00:01 |
|* 13 |    HASH JOIN           |           |   260K|    10M|  1148   (1)| 00:00:01 |
|* 14 |     TABLE ACCESS FULL  | TIMETABLE |  2869 | 40166 |    15   (0)| 00:00:01 |
|  15 |     TABLE ACCESS FULL  | IS_OPEN   |   932K|    24M|  1131   (1)| 00:00:01 |
-------------------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   1 - access("B"."BUSINESS_ID"="BUSINESS_ID")
   2 - access("HGFM"."GOODFORMEAL_ID"="GFM"."GOODFORMEAL_ID")
   3 - filter("GFM"."BRUNCH"=1)
   4 - access("B"."BUSINESS_ID"="HGFM"."BUSINESS_ID")
   5 - access("B"."BUSINESS_ID"="MEDIAN_B"."BUSINESS_ID")
   7 - filter(PERCENTILE_CONT(0.500000) WITHIN GROUP ( ORDER BY
           "R"."STARS")>=4.5)
  10 - filter("B"."IS_OPEN"=1)
  13 - access("IO"."TIMETABLE_ID"="T"."TIMETABLE_ID")
  14 - filter("T"."DAY"='Saturday' OR "T"."DAY"='Sunday')
Note
-----
   - this is an adaptive plan
```

DIAS: Data-Intensive Applications and Systems Laboratory
School of Computer and Communication Sciences
Ecole Polytechnique Fédérale de Lausanne
Building BC, Station 14
CH-1015 Lausanne
URL: http://dias.epfl.ch/

Improved plan:

```
-------------------------------------------------------------------------------------------------
| Id  | Operation                | Name                 | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT         |                      |   539 | 67914 |  3659   (2)| 00:00:01 |
|*  1 |  HASH JOIN SEMI          |                      |   539 | 67914 |  3659   (2)| 00:00:01 |
|*  2 |   HASH JOIN              |                      |   586 | 59772 |  2510   (2)| 00:00:01 |
|*  3 |    TABLE ACCESS FULL     | GOODFORMEAL          |    30 |   180 |     3   (0)| 00:00:01 |
|*  4 |    HASH JOIN             |                      |  1231 |   115K|  2507   (2)| 00:00:01 |
|*  5 |     HASH JOIN            |                      |  6565 |   448K|  2438   (2)| 00:00:01 |
|   6 |      VIEW                |                      |  6565 |   153K|  1310   (2)| 00:00:01 |
|*  7 |       FILTER             |                      |       |       |            |          |
|   8 |        SORT GROUP BY     |                      |  6565 |   166K|  1310   (2)| 00:00:01 |
|   9 |         INDEX FAST FULL SCAN| REVIEW_BUSINESID_STARS |  918K|    22M|  1288   (1)| 00:00:01 |
|* 10 |      TABLE ACCESS FULL   | BUSINESS             |   158K|  7121K|  1128   (1)| 00:00:01 |
|  11 |     TABLE ACCESS FULL    | HAS_GFM              | 29714 |   754K|    68   (0)| 00:00:01 |
|  12 |    VIEW                  | VW_NSO_1             |   260K|  6112K|  1148   (1)| 00:00:01 |
|* 13 |     HASH JOIN            |                      |   260K|    10M|  1148   (1)| 00:00:01 |
|* 14 |      TABLE ACCESS FULL   | TIMETABLE            |  2869 | 40166 |    15   (0)| 00:00:01 |
|  15 |      TABLE ACCESS FULL   | IS_OPEN              |   932K|    24M|  1131   (1)| 00:00:01 |
-------------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   1 - access("B"."BUSINESS_ID"="BUSINESS_ID")
   2 - access("HGFM"."GOODFORMEAL_ID"="GFM"."GOODFORMEAL_ID")
   3 - filter("GFM"."BRUNCH"=1)
   4 - access("B"."BUSINESS_ID"="HGFM"."BUSINESS_ID")
   5 - access("B"."BUSINESS_ID"="MEDIAN_B"."BUSINESS_ID")
   7 - filter(PERCENTILE_CONT(0.500000) WITHIN GROUP ( ORDER BY "R"."STARS")>=4.5)
  10 - filter("B"."IS_OPEN"=1)
  13 - access("IO"."TIMETABLE_ID"="T"."TIMETABLE_ID")
  14 - filter("T"."DAY"='Saturday' OR "T"."DAY"='Sunday')

Note
-----
   - this is an adaptive plan
```

11:10

DIAS: Data-Intensive Applications and Systems Laboratory
School of Computer and Communication Sciences
Ecole Polytechnique Fédérale de Lausanne
Building BC, Station 14
CH-1015 Lausanne
URL: http://dias.epfl.ch/

Query 16:
Initial runtime: 2.142s

Optimized running time: 0.128s

Explain the improvement: We don't need to scan the complete Review table (cost = 6062) anymore, we do a index range scan (cost = 2)

Initial plan:

```
-----------------------------------------------------------------------------------------------------------
| Id  | Operation                                | Name                | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                         |                     |     1 |    54 |  7275   (1)| 00:00:01 |
|*  1 |  HASH JOIN                               |                     |     1 |    54 |  7275   (1)| 00:00:01 |
|   2 |   VIEW                                   | VW_NSO_1            |     1 |     3 |  6146   (1)| 00:00:01 |
|   3 |    HASH UNIQUE                           |                     |     1 |       |  6146   (1)| 00:00:01 |
|*  4 |     COUNT STOPKEY                        |                     |       |       |            |          |
|   5 |      VIEW                                |                     |     1 |       |  6145   (1)| 00:00:01 |
|*  6 |       SORT ORDER BY STOPKEY              |                     |     1 |   149 |  6145   (1)| 00:00:01 |
|   7 |        HASH GROUP BY                     |                     |     1 |   149 |  6145   (1)| 00:00:01 |
|   8 |         NESTED LOOPS                     |                     |     1 |   149 |  6143   (1)| 00:00:01 |
|   9 |          NESTED LOOPS                    |                     |     1 |   149 |  6143   (1)| 00:00:01 |
|  10 |           NESTED LOOPS                   |                     |     1 |   133 |  6142   (1)| 00:00:01 |
|  11 |            NESTED LOOPS                  |                     |     1 |   124 |  6141   (1)| 00:00:01 |
|* 12 |             HASH JOIN                    |                     |     1 |    97 |  6139   (1)| 00:00:01 |
|  13 |              VIEW                        | VW_GBC_7            |    52 |  2548 |  6072   (1)| 00:00:01 |
|  14 |               HASH GROUP BY              |                     |    52 |  3172 |  6072   (1)| 00:00:01 |
|* 15 |                HASH JOIN                 |                     |   110 |  6710 |  6071   (1)| 00:00:01 |
|  16 |                 MERGE JOIN               |                     |    16 |   560 |     6  (17)| 00:00:01 |
|* 17 |                  TABLE ACCESS BY INDEX ROWID| DIETARYRESTRICTIONS |     3 |    27 |     2   (0)| 00:00:01 |
|  18 |                   INDEX FULL SCAN        | SYS_C00112639       |    10 |       |     1   (0)| 00:00:01 |
|* 19 |                  SORT JOIN               |                     |    52 |  1352 |     4  (25)| 00:00:01 |
|  20 |                   TABLE ACCESS FULL      | HAS_DR              |    52 |  1352 |     3   (0)| 00:00:01 |
|  21 |                 TABLE ACCESS FULL        | REVIEW              |  918K |   22M |  6062   (1)| 00:00:01 |
|  22 |               NESTED LOOPS               |                     |  2332 |  109K |    67   (0)| 00:00:01 |
|* 23 |                TABLE ACCESS FULL         | TIMETABLE           |    26 |   546 |    15   (0)| 00:00:01 |
|* 24 |                INDEX RANGE SCAN          | SYS_C00112533       |    91 |  2457 |     2   (0)| 00:00:01 |
|  25 |              TABLE ACCESS BY INDEX ROWID | IS_IN               |     1 |    27 |     2   (0)| 00:00:01 |
|* 26 |               INDEX UNIQUE SCAN          | SYS_C00114413       |     1 |       |     1   (0)| 00:00:01 |
|  27 |             TABLE ACCESS BY INDEX ROWID  | POSTAL_CODE         |     1 |     9 |     1   (0)| 00:00:01 |
|* 28 |              INDEX UNIQUE SCAN           | SYS_C00114410       |     1 |       |     0   (0)| 00:00:01 |
|* 29 |            INDEX UNIQUE SCAN             | SYS_C00114426       |     1 |       |     0   (0)| 00:00:01 |
|* 30 |           TABLE ACCESS BY INDEX ROWID    | CITY                |     1 |    16 |     1   (0)| 00:00:01 |
|  31 |    TABLE ACCESS FULL                     | BUSINESS            |  192K |  9592K|  1128   (1)| 00:00:01 |
-----------------------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   1 - access("1"=TO_NUMBER("B2"."BUSINESS_ID"))
   4 - filter(ROWNUM<=5)
   6 - filter(ROWNUM<=5)
  12 - access("ITEM_1"="IO"."BUSINESS_ID")
  15 - access("R"."BUSINESS_ID"="HD"."BUSINESS_ID")
  17 - filter("D"."VEGETARIAN"=1 AND "D"."VEGAN"=1)
  19 - access("HD"."DIETARYRESTRICTIONS_ID"="D"."DIETARYRESTRICTIONS_ID")
       filter("HD"."DIETARYRESTRICTIONS_ID"="D"."DIETARYRESTRICTIONS_ID")
  23 - filter(TO_CHAR(INTERNAL_FUNCTION("T"."OPENING"),'HH24:MI')<='14:00' AND
              TO_CHAR(INTERNAL_FUNCTION("T"."CLOSING"),'HH24:MI')>='16:00')
  24 - access("IO"."TIMETABLE_ID"="T"."TIMETABLE_ID")
  26 - access("ITEM_1"="II"."BUSINESS_ID")
  28 - access("II"."POSTAL_CODE_ID"="PC"."POSTAL_CODE_ID")
  29 - access("PC"."CITY_ID"="C"."CITY_ID")
  30 - filter("C"."CITY"='los angeles')
```

DIAS: Data-Intensive Applications and Systems Laboratory
School of Computer and Communication Sciences
Ecole Polytechnique Fédérale de Lausanne
Building BC, Station 14
CH-1015 Lausanne
URL: http://dias.epfl.ch/

Improved plan:

```
-----------------------------------------------------------------------------------------------------------------------
| Id  | Operation                                | Name                    | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                         |                         |     1 |    54 |  1246   (1)| 00:00:01 |
|*  1 |  HASH JOIN                               |                         |     1 |    54 |  1246   (1)| 00:00:01 |
|   2 |   VIEW                                   | VW_NSO_1                |     1 |     3 |   117   (5)| 00:00:01 |
|   3 |    HASH UNIQUE                           |                         |     1 |       |   117   (5)| 00:00:01 |
|*  4 |     COUNT STOPKEY                        |                         |       |       |            |          |
|   5 |      VIEW                                |                         |     1 |       |   116   (4)| 00:00:01 |
|*  6 |       SORT ORDER BY STOPKEY              |                         |     1 |   149 |   116   (4)| 00:00:01 |
|   7 |        HASH GROUP BY                     |                         |     1 |   149 |   116   (4)| 00:00:01 |
|   8 |         NESTED LOOPS                     |                         |     1 |   149 |   114   (2)| 00:00:01 |
|   9 |          NESTED LOOPS                    |                         |     2 |   149 |   114   (2)| 00:00:01 |
|  10 |           NESTED LOOPS                   |                         |     2 |   266 |   112   (2)| 00:00:01 |
|  11 |            NESTED LOOPS                  |                         |     2 |   248 |   110   (2)| 00:00:01 |
|* 12 |             HASH JOIN                    |                         |     2 |   194 |   106   (2)| 00:00:01 |
|  13 |              VIEW                        | VW_GBC_7                |   110 |  5390 |    39   (6)| 00:00:01 |
|  14 |               HASH GROUP BY              |                         |   110 |  6710 |    39   (6)| 00:00:01 |
|  15 |                NESTED LOOPS              |                         |   110 |  6710 |    38   (3)| 00:00:01 |
|  16 |                 MERGE JOIN               |                         |    16 |   560 |     6  (17)| 00:00:01 |
|* 17 |                  TABLE ACCESS BY INDEX ROWID| DIETARYRESTRICTIONS   |     3 |    27 |     2   (0)| 00:00:01 |
|  18 |                   INDEX FULL SCAN        | SYS_C00112639           |    10 |       |     1   (0)| 00:00:01 |
|* 19 |                  SORT JOIN               |                         |    52 |  1352 |     4  (25)| 00:00:01 |
|  20 |                   TABLE ACCESS FULL      | HAS_DR                  |    52 |  1352 |     3   (0)| 00:00:01 |
|* 21 |                 INDEX RANGE SCAN         | REVIEW_BUSINESID_STARS  |     7 |   182 |     2   (0)| 00:00:01 |
|  22 |               NESTED LOOPS               |                         |  2332 |  109K |    67   (0)| 00:00:01 |
|* 23 |                TABLE ACCESS FULL         | TIMETABLE               |    26 |   546 |    15   (0)| 00:00:01 |
|* 24 |                INDEX RANGE SCAN          | SYS_C00112533           |    91 |  2457 |     2   (0)| 00:00:01 |
|  25 |              TABLE ACCESS BY INDEX ROWID | IS_IN                   |     1 |    27 |     2   (0)| 00:00:01 |
|* 26 |               INDEX UNIQUE SCAN          | SYS_C00114413           |     1 |       |     1   (0)| 00:00:01 |
|  27 |             TABLE ACCESS BY INDEX ROWID  | POSTAL_CODE             |     1 |     9 |     1   (0)| 00:00:01 |
|* 28 |              INDEX UNIQUE SCAN           | SYS_C00114410           |     1 |       |     0   (0)| 00:00:01 |
|* 29 |            INDEX UNIQUE SCAN             | SYS_C00114426           |     1 |       |     0   (0)| 00:00:01 |
|* 30 |           TABLE ACCESS BY INDEX ROWID    | CITY                    |     1 |    16 |     1   (0)| 00:00:01 |
|  31 |   TABLE ACCESS FULL                      | BUSINESS                |  192K |  9592K|  1128   (1)| 00:00:01 |
-----------------------------------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   1 - access("1"=TO_NUMBER("B2"."BUSINESS_ID"))
   4 - filter(ROWNUM<=5)
   6 - filter(ROWNUM<=5)
  12 - access("ITEM_1"="IO"."BUSINESS_ID")
  17 - filter("D"."VEGETARIAN"=1 AND "D"."VEGAN"=1)
  19 - access("HD"."DIETARYRESTRICTIONS_ID"="D"."DIETARYRESTRICTIONS_ID")
       filter("HD"."DIETARYRESTRICTIONS_ID"="D"."DIETARYRESTRICTIONS_ID")
  21 - access("R"."BUSINESS_ID"="HD"."BUSINESS_ID")
  23 - filter(TO_CHAR(INTERNAL_FUNCTION("T"."OPENING"),'HH24:MI')<='14:00' AND
              TO_CHAR(INTERNAL_FUNCTION("T"."CLOSING"),'HH24:MI')>='16:00')
  24 - access("IO"."TIMETABLE_ID"="T"."TIMETABLE_ID")
  26 - access("ITEM_1"="II"."BUSINESS_ID")
  28 - access("II"."POSTAL_CODE_ID"="PC"."POSTAL_CODE_ID")
  29 - access("PC"."CITY_ID"="C"."CITY_ID")
  30 - filter("C"."CITY"='los angeles')
```

DIAS: Data-Intensive Applications and Systems Laboratory
School of Computer and Communication Sciences
Ecole Polytechnique Fédérale de Lausanne
Building BC, Station 14
CH-1015 Lausanne
URL: http://dias.epfl.ch/

Query 19:
Initial runtime: 31.06s

Optimized running time: 0.314s

Second Optimized running time: 0.247s

Explain the improvement: The plans are very different, one huge improvement come from the fact that we do not have a double nested loop anymore.
Furthermore the two similar sub query are merged into one which half the computations. The filter is much lighter in the new version because the joins are more restrictive. (more explanations in optimization step 2)

```
-----------------------------------------------------------------------------------------------------
| Id  | Operation                      | Name          | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT               |               |  1124 | 17984 |   601   (1)| 00:00:01 |
|*  1 |  FILTER                        |               |       |       |            |          |
|   2 |   TABLE ACCESS FULL            | CITY          |  1124 | 17984 |     3   (0)| 00:00:01 |
|   3 |   SORT AGGREGATE               |               |     1 |    26 |            |          |
|*  4 |    VIEW                        |               |   100 |  2600 |   598   (1)| 00:00:01 |
|*  5 |     WINDOW SORT PUSHED RANK    |               |   170 | 10710 |   598   (1)| 00:00:01 |
|   6 |      NESTED LOOPS              |               |   170 | 10710 |   597   (1)| 00:00:01 |
|   7 |       NESTED LOOPS             |               |   170 | 10710 |   597   (1)| 00:00:01 |
|*  8 |        HASH JOIN               |               |   170 |  6120 |   257   (1)| 00:00:01 |
|*  9 |         TABLE ACCESS FULL      | POSTAL_CODE   |    18 |   162 |    17   (0)| 00:00:01 |
|  10 |         TABLE ACCESS FULL      | IS_IN         |  192K |  5078K|   239   (1)| 00:00:01 |
|* 11 |        INDEX UNIQUE SCAN       | SYS_C00110565 |     1 |       |     1   (0)| 00:00:01 |
|  12 |       TABLE ACCESS BY INDEX ROWID| BUSINESS    |     1 |    27 |     2   (0)| 00:00:01 |
|  13 |   SORT AGGREGATE               |               |     1 |    26 |            |          |
|* 14 |    VIEW                        |               |   170 |  4420 |   598   (1)| 00:00:01 |
|  15 |     WINDOW SORT                |               |   170 | 10710 |   598   (1)| 00:00:01 |
|  16 |      NESTED LOOPS              |               |   170 | 10710 |   597   (1)| 00:00:01 |
|  17 |       NESTED LOOPS             |               |   170 | 10710 |   597   (1)| 00:00:01 |
|* 18 |        HASH JOIN               |               |   170 |  6120 |   257   (1)| 00:00:01 |
|* 19 |         TABLE ACCESS FULL      | POSTAL_CODE   |    18 |   162 |    17   (0)| 00:00:01 |
|  20 |         TABLE ACCESS FULL      | IS_IN         |  192K |  5078K|   239   (1)| 00:00:01 |
|* 21 |        INDEX UNIQUE SCAN       | SYS_C00110565 |     1 |       |     1   (0)| 00:00:01 |
|  22 |       TABLE ACCESS BY INDEX ROWID| BUSINESS    |     1 |    27 |     2   (0)| 00:00:01 |
-----------------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   1 - filter( (SELECT SUM("R1"."REVIEW_COUNT") FROM  (SELECT "B1"."REVIEW_COUNT"
             "REVIEW_COUNT",ROW_NUMBER() OVER ( ORDER BY INTERNAL_FUNCTION("B1"."REVIEW_COUNT") DESC )
             "RN" FROM "POSTAL_CODE" "PC1","IS_IN" "II1","BUSINESS" "B1" WHERE
             "B1"."BUSINESS_ID"="II1"."BUSINESS_ID" AND "II1"."POSTAL_CODE_ID"="PC1"."POSTAL_CODE_ID"
             AND "PC1"."CITY_ID"=:B1) "R1" WHERE "R1"."RN"<=100)>=2*NVL( (SELECT
             SUM("R2"."REVIEW_COUNT") FROM  (SELECT "B"."REVIEW_COUNT" "REVIEW_COUNT",ROW_NUMBER()
             OVER ( ORDER BY INTERNAL_FUNCTION("B"."REVIEW_COUNT") DESC ) "RN" FROM "POSTAL_CODE"
             "PC","IS_IN" "II","BUSINESS" "B" WHERE "B"."BUSINESS_ID"="II"."BUSINESS_ID" AND
             "II"."POSTAL_CODE_ID"="PC"."POSTAL_CODE_ID" AND "PC"."CITY_ID"=:B2) "R2" WHERE
             "R2"."RN">100),0))
   4 - filter("R1"."RN"<=100)
   5 - filter(ROW_NUMBER() OVER ( ORDER BY INTERNAL_FUNCTION("B1"."REVIEW_COUNT") DESC
             )<=100)
   8 - access("II1"."POSTAL_CODE_ID"="PC1"."POSTAL_CODE_ID")
   9 - filter("PC1"."CITY_ID"=:B1)
  11 - access("B1"."BUSINESS_ID"="II1"."BUSINESS_ID")
  14 - filter("R2"."RN">100)
  18 - access("II"."POSTAL_CODE_ID"="PC"."POSTAL_CODE_ID")
  19 - filter("PC"."CITY_ID"=:B1)
  21 - access("B"."BUSINESS_ID"="II"."BUSINESS_ID")
```

DIAS: Data-Intensive Applications and Systems Laboratory
School of Computer and Communication Sciences
Ecole Polytechnique Fédérale de Lausanne
Building BC, Station 14
CH-1015 Lausanne
URL: http://dias.epfl.ch/

Optimization step 1:

The new query design:

```
SELECT c.city
FROM City c,
   (SELECT SUM(case when r.rn<100 then r.review_count end) first100,NVL(SUM(case when r.rn>100 then
r.review_count end),0) after100,r.city_id
   FROM (SELECT b.review_count ,pc.city_id ,ROW_NUMBER() OVER(PARTITION BY pc.city_id ORDER BY
b.review_count DESC) as rn
      FROM Business b, Is_in ii, Postal_code pc
      WHERE b.business_id=ii.business_id AND ii.postal_code_id=pc.postal_code_id) r
   GROUP BY r.city_id) r2
WHERE c.city_id= r2.city_id and r2.first100>=2*r2.after100
```

Query explanation:

I will first explain the subsubquery "r", which create for each business a tuple containing the number of review and its rank among the other business in the same city

Then the sub query r2 yield for each city, a sum of the review_count of the top100Business and the sum for the out-of-top100 business

Then I simply keep the city for which the first sum is at least twice the second sum and display its name

```
-------------------------------------------------------------------------------------------------
| Id  | Operation                | Name        | Rows  | Bytes |TempSpc| Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT         |             |    57 |  1653 |       |  5000   (1)| 00:00:01 |
|*  1 |  HASH JOIN               |             |    57 |  1653 |       |  5000   (1)| 00:00:01 |
|   2 |   VIEW                   |             |    57 |   741 |       |  4997   (1)| 00:00:01 |
|*  3 |    FILTER                |             |       |       |       |            |          |
|   4 |     HASH GROUP BY        |             |    57 |  1710 |       |  4997   (1)| 00:00:01 |
|   5 |      VIEW                |             |   191K|  5608K|       |  4997   (1)| 00:00:01 |
|   6 |       WINDOW SORT        |             |   191K|    11M|    13M|  4997   (1)| 00:00:01 |
|*  7 |        HASH JOIN         |             |   191K|    11M|       |  2098   (1)| 00:00:01 |
|   8 |         TABLE ACCESS FULL| POSTAL_CODE | 20117 |   176K|       |    17   (0)| 00:00:01 |
|*  9 |         HASH JOIN        |             |   192K|     9M|  7336K|  2080   (1)| 00:00:01 |
|  10 |          TABLE ACCESS FULL| BUSINESS    |   192K|  5078K|       |  1128   (1)| 00:00:01 |
|  11 |          TABLE ACCESS FULL| IS_IN       |   192K|  5078K|       |   239   (1)| 00:00:01 |
|  12 |   TABLE ACCESS FULL      | CITY        |  1124 | 17984 |       |     3   (0)| 00:00:01 |
-------------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   1 - access("C"."CITY_ID"="R2"."CITY_ID")
   3 - filter(SUM(CASE  WHEN "R"."RN"<100 THEN "R"."REVIEW_COUNT" END )>=2*NVL(SUM(CASE
              WHEN "R"."RN">100 THEN "R"."REVIEW_COUNT" END ),0))
   7 - access("II"."POSTAL_CODE_ID"="PC"."POSTAL_CODE_ID")
   9 - access("B"."BUSINESS_ID"="II"."BUSINESS_ID")

Note
-----
   - this is an adaptive plan
```

DIAS: Data-Intensive Applications and Systems Laboratory
School of Computer and Communication Sciences
Ecole Polytechnique Fédérale de Lausanne
Building BC, Station 14
CH-1015 Lausanne
URL: http://dias.epfl.ch/

Optimization step 2:

Explain the improvement:
The I/O cost before the creation of the index
5000+5000+4997+4997+4997+4997+2098+17+2080+1128+239+3=35553
The I/O cost after the creation of the index
4143+4143+4140+4140+4140+4140+1241+17+1224+272+239+3=27842
The ratio
27842/35553=0.783
Being able to iterate over the businesses by their number of review_count allow us to retrieve the information we need much faster, instead of doing a full scan of the business table.
If we look at the Operation 10, it takes 1128 I/O in the first version whereas the second version only need 272 I/O

```
CREATE INDEX business_reviewcount_id
ON business(review_count,business_id)
```

```
-----------------------------------------------------------------------------------------------------------------
| Id  | Operation                    | Name                    | Rows  | Bytes |TempSpc| Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |                         |    57 |  1653 |       |  4143   (1)| 00:00:01 |
|*  1 |  HASH JOIN                   |                         |    57 |  1653 |       |  4143   (1)| 00:00:01 |
|   2 |   VIEW                       |                         |    57 |   741 |       |  4140   (1)| 00:00:01 |
|*  3 |    FILTER                    |                         |       |       |       |            |          |
|   4 |     HASH GROUP BY            |                         |    57 |  1710 |       |  4140   (1)| 00:00:01 |
|   5 |      VIEW                    |                         |   191K|  5608K|       |  4140   (1)| 00:00:01 |
|   6 |       WINDOW SORT            |                         |   191K|    11M|    13M|  4140   (1)| 00:00:01 |
|*  7 |        HASH JOIN             |                         |   191K|    11M|       |  1241   (1)| 00:00:01 |
|   8 |         TABLE ACCESS FULL    | POSTAL_CODE             | 20117 |   176K|       |    17   (0)| 00:00:01 |
|*  9 |         HASH JOIN            |                         |   192K|     9M|  7336K|  1224   (1)| 00:00:01 |
|  10 |          INDEX FAST FULL SCAN| BUSINESS_REVIEWCOUNT_ID |   192K|  5078K|       |   272   (1)| 00:00:01 |
|  11 |          TABLE ACCESS FULL   | IS_IN                   |   192K|  5078K|       |   239   (1)| 00:00:01 |
|  12 |   TABLE ACCESS FULL          | CITY                    |  1124 | 17984 |       |     3   (0)| 00:00:01 |
-----------------------------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   1 - access("C"."CITY_ID"="R2"."CITY_ID")
   3 - filter(SUM(CASE  WHEN "R"."RN"<100 THEN "R"."REVIEW_COUNT" END )>=2*NVL(SUM(CASE  WHEN
              "R"."RN">100 THEN "R"."REVIEW_COUNT" END ),0))
   7 - access("II"."POSTAL_CODE_ID"="PC"."POSTAL_CODE_ID")
   9 - access("B"."BUSINESS_ID"="II"."BUSINESS_ID")

Note
-----
   - this is an adaptive plan
```

DIAS: Data-Intensive Applications and Systems Laboratory
School of Computer and Communication Sciences
Ecole Polytechnique Fédérale de Lausanne
Building BC, Station 14
CH-1015 Lausanne
URL: http://dias.epfl.ch/

General Comments

For this last deliverable, we've tried to parallelize the work and we divided the queries into three: we each wrote the corresponding SQL queries. Once it was done, we sent each other our queries for reviews and potential optimizations. The index optimizations were quite challenging to find, with our ER model some were not very suitable for indexing.