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 2019

Team No: 22

Names: Gerald Sula, Georgios Fotiadis, Ridha Chahed

Contents

Contents 1
Deliverable 1 2
Assumptions 2
Entity Relationship Schema 2
Schema 3
Description 3
Relational Schema 4
ER schema to Relational schema 6
DDL 6
General Comments 6
Deliverable 2 10
Assumptions 10
Data Loading 10
Query Implementation 10
Query a: 10
Description of logic: 10
SQL statement 10
Interface 10
Design logic Description 10
Screenshots 10
General Comments 10
Deliverable 3 11

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/

Assumptions 11
Query Implementation 11
Query a: 11
Description of logic: 11
SQL statement 11
Query Analysis 11
Selected Queries (and why) 11
Query 1 11
Query 2 11
Query 3 11
Interface 12
Design logic Description 12
Screenshots 12
General Comments 12

Deliverable 1

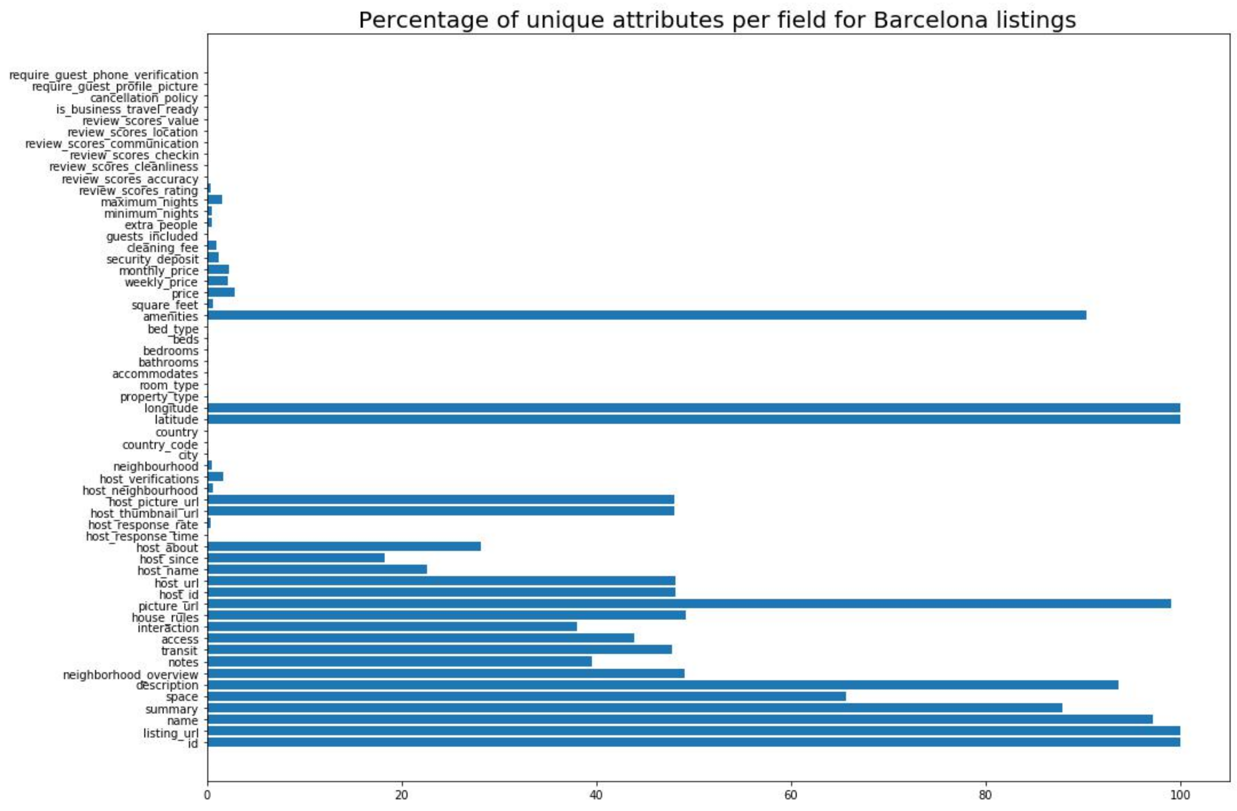
Assumptions

We are using as a central reference point the Listing entity (which has only 3 attributes unique to each listing) because almost all of the data could be linked to it in a very intuitive way.

All the entities have been selected judging on the principle that: if the attributes found in one instance of an entity are repeated in the data, those attributes must be included in one entity. We have made sure to "package" contextually similar attributes into one entity, as to not increase the complexity of the schema. For the attributes which can be different from one instance to the other, we have set them as relationship attributes (to avoid having a new entity all together, and also for not violating the aforementioned principle).

This is an overview of the percentage of unique attributes per field of the Barcelona listings dataset. Note that the plot is exactly the same for every 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/

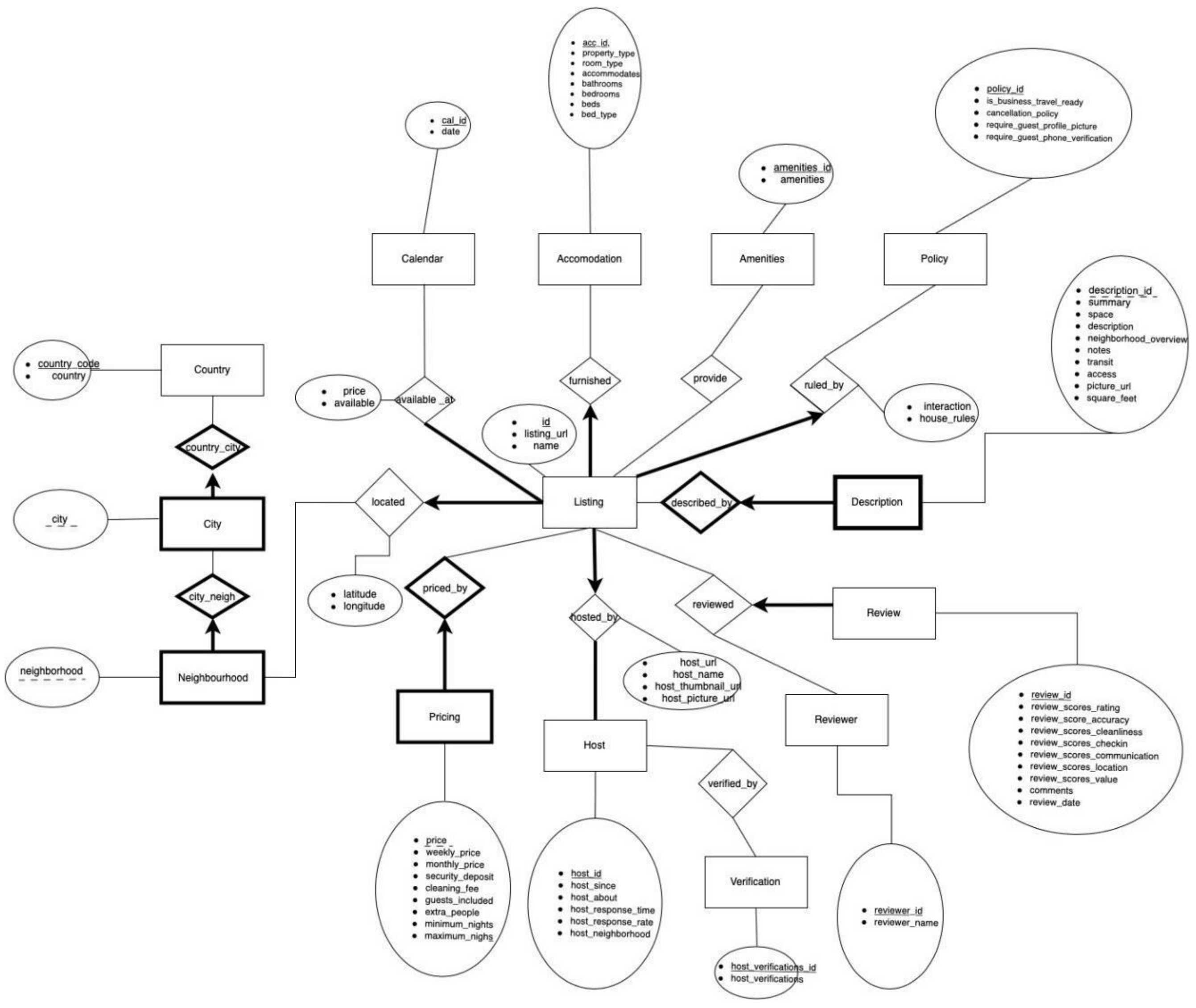


Entity Relationship Schema

Schema

Note: We included all the attributes of an entity in a big bubble as to conserve space in the diagram.

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

<u>**Calendar**</u>: there is a 'one to many' relation with listing because one listing will have at least one date of availability. The 'price' and 'available' attributes were set as relational attributes, because they were different for different listings (as they should be).

<u>**Accommodation**</u>: This entity describes the interior of the listing's apartment. All of these attributes will change from listing to listing so getting them together in one entity seems to be the right choice. Every listing will have exactly one set of those attributes, hence we used the 'exactly one' relation. It can happen that there are more than one listing that have the same set of attributes of this kind, that's why we opted out on having a weak relation here.

Amenities: The data contained here would make sense to be included in the accommodation entity, but since listings have more than exactly one amenity, we decided to create a separate simple entity.

Policy: Similar to the accommodation entity, we grouped together attributes which are unique to the listing (and also describe the same thing, in this case the rules of the listing) into one place. There is again an 'exactly one' relationship for the same reasons. One difference is that there are two relationship attributes here (interaction and house_rules). We decided to put them here because when looking at the data we saw that the policy was similar in most of the listings, with the exception of these two fields, as you can see in the plot above.

Description: Once again we grouped together attributes which describe the same aspect of the listing. We made this a weak entity, because a description should not exist on its own, it should always be paired with a listing.

Review and Reviewer: We decided to go with a ternary relation here between review-reviewer-listing. This is because they will always be used together: A review is given by a reviewer for a listing. The attributes we assigned to each entity are self-explanatory and there is an 'exactly one" relation for review because that has to be unique to the listing-reviewer. There are 'many to many' relations for both listing and reviewer because a listing can have many reviews from different reviewers, and a reviewer can leave many reviews for different listings.

Host: The host entity has all the attributes which describe a host. There is an "exactly one" relation from listing, because a listing should have exactly one host (we are not allowing multiple host to manage the same listing, same as Airbnb does in real life). The host has an 'at least' relation with listing because it makes no sense to be a host when you have no listing to manage.

Verification: We decided to split the verification into a new entity because the data of the field 'host_verification' was reused multiple times for different hosts. There is an 'exactly one' relation with hos since every host has only one verification. Pricing: For the exact same reasons as with description we decided do have a weak entity of similar data here.

Country, City, Neighborhood: Here we are using the logic that a Country can have multiple Cities, and each City can have multiple Neighborhoods. City and Neighborhood are weak entities to the ones above them because, for example, there can be no city which has no country. Finally, we 'connect' the most descriptive of the three (Neighborhood) with Listing with an 'exactly one' relation, because each listing is located at exactly one neighborhood. The attributes of country, city and neighborhood are reused for different instances of listings, the ones that change (latitude and longitude), we put inside the relationship, as to be able to keep the property of reusability.

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, DDL:

```
CREATE TABLE Calendar(cal_id CHAR(32),
              date DATE,
              PRIMARY KEY (cal_id))

CREATE TABLE available_at(id CHAR(32),
                cal_id CHAR(32),
                price FLOAT,
                available CHAR(1),
                PRIMARY KEY (id,cal_id),
                FOREIGN KEY (cal_id) REFERENCES Calendar(cal_id),
                FOREIGN KEY (id) REFERENCES Listing(id)
                /*can't ensure participation constraint*/)

CREATE TABLE Accomodation(acc_id CHAR(32),
                property_type CHAR(32),
                room_type CHAR(32),
                accomodates CHA(32),
                bathrooms INTEGER,
                bedrooms INTEGER,
                beds INTEGER,
                bed_type CHAR(32,
                PRIMARY KEY (acc_id) )

CREATE TABLE Amenities(amenities_id CHAR(32),
               amenities CHAR(32))

CREATE TABLE Provide(id CHAR(32),
             amenities_id CHAR(32),
             PRIMARY KEY (id,amenities_id),
             FOREIGN KEY (amenities_id) REFERENCES Amenities(amenities_id),
             FOREIGN KEY (id) REFERENCES Listing(id) )

CREATE TABLE Policy(policy_id CHAR(32),
             is_buisness_travel_ready CHAR(1),
                cancellation_policy CHAR(32),
                require_guest_profile_picture CHAR(1),
                require_guest_phone_verification CHAR(1)
                PRIMARY KEY (policy_id) )

  CREATE TABLE Described_Description(description_id CHAR(32),
                         summary CHAR(1024),
                         space CHAR(1024),
                         description CHAR(1024),
                         neighborhood_overview CHAR(1024),
                         notes CHAR(1024),
                         transit CHAR(1024),
                         access CHAR(1024),
                         picture URL CHAR(32),
                         square_feet FLOAT,
                         id CHAR(32) NOT NULL
                         PRIMARY KEY (description_id,id)
                         FOREIGN KEY (id) REFERENCES Listing(id)
                         ON DELETE CASCADE) )

CREATE TABLE Review(review_id CHAR (32),
                review_scores_rating INTEGER,
                review_score_accuracy INTEGER,
                review_scores_cleanliness INTEGER,
                review_scores_checkin INTEGER,
                review_scores_communication INTEGER,
                review_scores_location INTEGER,
                review_scores_value INTEGER,
                comments CHAR(1024),
                review_date DATE,
                id CHAR(32) NOT NULL,
                reviewer_id CHAR(32) NOT NULL,
                PRIMARY KEY (review_id),
                FOREIGN KEY (id) REFERENCES Listing(id),
                FOREIGN KEY (reviewer_id) REFERENCES Reviewer(reviewer_id) )

CREATE TABLE Reviewer(reviewer_id CHAR(32),
            reviewer_name CHAR(32)
            PRIMARY KEY (reviewer_id) )

CREATE TABLE Host(host_id CHAR(32),
          host_since DATE,
          host_about CHAR(1024),
          host_response_time FLOAT,
          host_response_rate FLOAT,
          host_neighborhood CHAR(32),
          PRIMARY KEY(host_id) )

CREATE TABLE Verification(host_verifications_id CHAR(32),
              host_verifications CHAR(32),
              PRIMARY KEY (host_verifications_id) )

CREATE TABLE Verified_by(host_id CHAR(32),
             host_verifications_id CHAR(32),
             PRIMARY KEY (host_id,host_verifications_id),
             FOREIGN KEY (host_id) REFERENCES Host(host_id),
             FOREIGN KEY (host_verifications_id) REFERENCES
             Verification(host_verifications_id) )

CREATE TABLE Pricing(price FLOAT,
           weekly_price FLOAT,
           monthly_price FLOAT,
           security_deposit FLOAT,
           cleaning_fee FLOAT,
           guests_included INTEGER,
           extra_people FLOAT,
           minimum_nights INTEGER,
           maximum_nighs INTEGER,
           id CHAR(32) NOT NULL,
           PRIMARY KEY(price,id),
           FOREIGN KEY (id) REFERENCES Listing(id) ON DELETE CASCADE )

CREATE TABLE Country(country_code CHAR(32),
           country CHAR(32),
               PRIMARY KEY (country_code) )

CREATE TABLE City(city CHAR(32),
             country_code CHAR(32),
             PRIMARY KEY (country_code,city),
             FOREIGN KEY (country_code) REFERENCES Country(country_code) )

CREATE TABLE Neighbourhood(neighborhood CHAR(32),
                   city CHAR(32),
                   country_code CHAR(32),
                   PRIMARY KEY (country_code,city,neighborhood),
                   FOREIGN KEY (country_code) REFERENCES Country(country_code),
                   FOREIGN KEY (city) REFERENCES City(city) )

CREATE TABLE Listing(id CHAR(32),
               lisitng_url CHAR(32),
               name CHAR(32),
               acc_id CHAR(32) NOT NULL,
               policy_id CHAR(32) NOT NULL,
               host_id CHAR(32) NOT NULL,
               host_url CHAR(32),
               host_name CHAR(32),
               host_thumbnail_url CHAR(32),
               host_picture_ur CHAR(32),
               neighborhood CHAR(32) NOT NULL
               PRIMARY KEY (id),
               FOREIGN KEY (acc_id) REFERENCES Accomodation(acc_id),
               FOREIGN KEY (policy_id) REFERENCES Policy(policy_id),
               FOREIGN KEY (host_id) REFERENCES Host(host_id),
               FOREIGN KEY (neighborhood) REFERENCES Neighbourhood(neighborhood) )
```

General Comments

We all worked on the design of the ER schema, for the rest we decided to split the workload like this:

- Ridha did the translation to the Relational Model
- George analysed the data to further optimize the ER schema for maximum data reusability and locality
- Gerald wrote the report explaining what each relation contains and justify our design choices

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

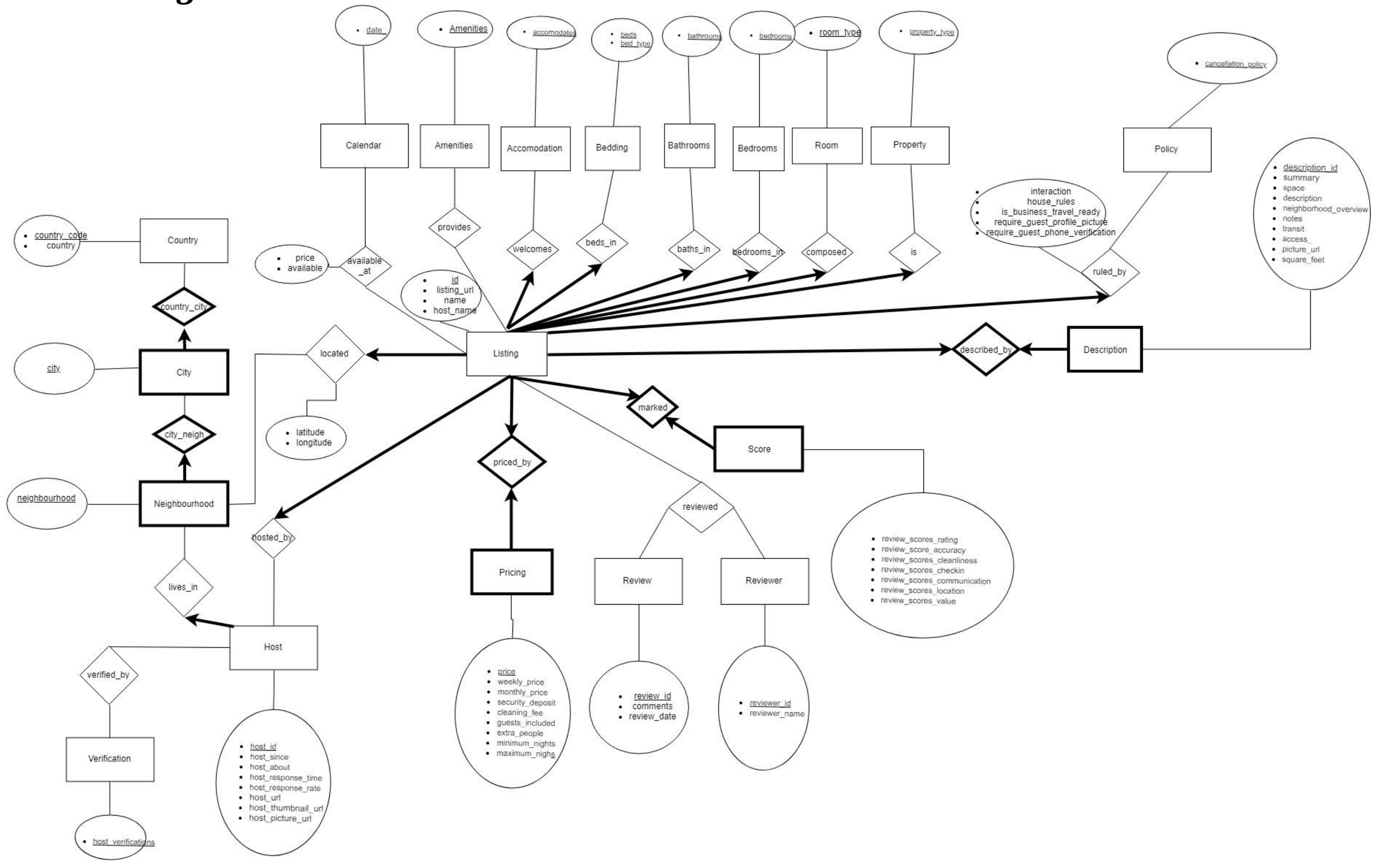
Changes from Deliverable 1:

Tables:

```
CREATE TABLE Calendar(date DATE,
            PRIMARY KEY (date));

CREATE TABLE available_at(id INTEGER,
              date DATE,
              price FLOAT,
              available CHAR(1),
              PRIMARY KEY (id,date),
              FOREIGN KEY (date) REFERENCES Calendar(date),
              FOREIGN KEY (id) REFERENCES Listing(id) )  ;

CREATE TABLE Accommodation(accommodates INTEGER,
              PRIMARY KEY (accomodates) ) ;

CREATE TABLE Bedding(beds INTEGER,
           bed_type CHAR(32),
           PRIMARY KEY (beds,bed_type) );

CREATE TABLE Bathrooms(bathrooms FLOAT,
            PRIMARY KEY (bathrooms) );

CREATE TABLE Bedrooms(bedrooms INTEGER,
            PRIMARY KEY(bedrooms) );

CREATE TABLE Room(room_type CHAR(32),
          PRIMARY KEY(room_type) );

CREATE TABLE Property(property_type CHAR(32),
            PRIMARY KEY (property_type) );

CREATE TABLE Amenities(amenities CHAR(32),
             PRIMARY KEY (amenities) );

CREATE TABLE Provides(id INTEGER,
           amenities CHAR(32),
           PRIMARY KEY (id,amenities),
           FOREIGN KEY (amenities) REFERENCES Amenities(amenities),
           FOREIGN KEY (id) REFERENCES Listing(id) ) ;
```

```
CREATE TABLE Policy(cancellation_policy CHAR(32),
                PRIMARY KEY (cancellation_policy) );

CREATE TABLE Described_Description(description_id INTEGER AUTO_INCREMENT,
                            summary VARCHAR(1024),
                            space VARCHAR(1024),
                            description VARCHAR(1024),
                            neighbourhood_overview VARCHAR(1024),
                            notes VARCHAR(1024),
                            transit VARCHAR(1024),
                            access VARCHAR(1024),
                            picture_URL CHAR(32),
                            square_feet FLOAT,
                            id INTEGER NOT NULL,
                            PRIMARY KEY (description_id,id),
                            FOREIGN KEY (id) REFERENCES Listing(id)
                            ON DELETE CASCADE) ;

CREATE TABLE Score (review_scores_rating INTEGER,
                review_score_accuracy INTEGER,
                review_scores_cleanliness INTEGER,
                review_scores_checkin INTEGER,
                review_scores_communication INTEGER,
                review_scores_location INTEGER,
                review_scores_value INTEGER,
                id INTEGER,
                PRIMARY KEY(id),
                FOREIGN KEY (id) REFERENCES Listing(id) ON DELETE CASCADE ) ;


CREATE TABLE Reviewer(reviewer_id INTEGER,
                 reviewer_name CHAR(32),
                 PRIMARY KEY (reviewer_id) )

CREATE TABLE Review (review_id INTEGER AUTO_INCREMENT,
                                                    review_date DATE,
                comments VARCHAR(1024),
                                                    PRIMARY key(review_id) )

CREATE TABLE Reviewed(id INTEGER,
                reviewer_id INTEGER,
                review_id INTEGER,
                PRIMARY KEY(id, review_id, reviewer_id),
                FOREIGN KEY (id) REFERENCES Listing(id),
                FOREIGN KEY (reviewer_id) REFERENCES Reviewer(reviewer_id),
                                                    FOREIGN KEY (review_id) REFERENCES Review(review_id) )

CREATE TABLE Host(host_id INTEGER,
              host_since DATE,
              host_about VARCHAR(1024),
              host_response_time FLOAT,
              host_response_rate FLOAT,
              host_neighborhood CHAR(32) NOT NULL,
                                                host_country_code INTEGER NOT NULL,
                                                host_city CHAR(32) NOT NULL,
              PRIMARY KEY(host_id),
            FOREIGN KEY (host_country_code,host_city,host_neighborhood) REFERENCES
Neighbourhood(country_code,city,neighbourhood) )

CREATE TABLE Verification(host_verifications CHAR(32),
                PRIMARY KEY (host_verifications) );

CREATE TABLE Verified_by(host_id INTEGER,
                host_verifications CHAR(32),
                PRIMARY KEY (host_id,host_verifications),
                FOREIGN KEY (host_id) REFERENCES Host(host_id),
                FOREIGN KEY (host_verifications) REFERENCES Verification(host_verifications) );

CREATE TABLE Pricing(price FLOAT,
             weekly_price FLOAT,
             monthly_price FLOAT,
             security_deposit FLOAT,
             cleaning_fee FLOAT,
             guests_included INTEGER,
             extra_people FLOAT,
             minimum_nights INTEGER,
             maximum_nighs INTEGER,
             id INTEGER NOT NULL,
             PRIMARY KEY(price,id),
             FOREIGN KEY (id) REFERENCES Listing(id) ON DELETE CASCADE ) ;

CREATE TABLE Country(country_code CHAR(2),
             country CHAR(32),
             PRIMARY KEY (country_code) ) ;

CREATE TABLE City(city CHAR(32),
            country_code CHAR(2),
            PRIMARY KEY (country_code,city),
            FOREIGN KEY (country_code) REFERENCES Country(country_code) );

CREATE TABLE Neighbourhood(neighbourhood CHAR(32),
                city CHAR(32),
                country_code CHAR(2),
                PRIMARY KEY (country_code,city,neighbourhood),
                FOREIGN KEY (country_code) REFERENCES Country(country_code),
                FOREIGN KEY (city) REFERENCES City(city) );

CREATE TABLE Listing(id INTEGER,
             listing_url CHAR(32),
             name CHAR(32),
             accommodates CHAR(32) NOT NULL,
             cancellation_policy CHAR(32) NOT NULL,
             host_id INTEGER NOT NULL,
             host_name CHAR(32),
             neighbourhood CHAR(32) NOT NULL,
                                        city CHAR(32) NOT NULL,
                                        country_code CHAR(2) NOT NULL,
             latitude FLOAT,
             longitude FLOAT,
             property_type CHAR(32) NOT NULL,
             room_type CHAR(32) NOT NULL,
             bathrooms FLOAT NOT NULL,
             bedrooms INTEGER NOT NULL,
             beds INTEGER NOT NULL,
             bed_type CHAR(32) NOT NULL,
                                        interaction VARCHAR(1024),
                                        house_rules VARCHAR(1024),
                                        is_business_travel_ready CHAR(1),
                                        require_guest_profile_picture CHAR(1),
                                        require_guest_phone_verification CHAR(1),
             PRIMARY KEY (id),
             FOREIGN KEY (property_type) REFERENCES Property(property_type),
             FOREIGN KEY (room_type) REFERENCES Room(room_type),
             FOREIGN KEY (accommodates) REFERENCES Accommodation(accommodates),
             FOREIGN KEY (bathrooms) REFERENCES Bathrooms(bathrooms),
             FOREIGN KEY (bedrooms) REFERENCES Bedrooms(bedrooms),
             FOREIGN KEY (beds,bed_type) REFERENCES Bedding(beds,bed_type),
             FOREIGN KEY (cancellation_policy) REFERENCES Policy(cancellation_policy),
             FOREIGN KEY (host_id) REFERENCES Host(host_id),
             FOREIGN KEY (country_code,city,neighbourhood) REFERENCES
Neighbourhood(country_code,city,neighbourhood),
                                        FOREIGN KEY (country_code,city) REFERENCES City(country_code,city),
                                        FOREIGN KEy (country_code) REFERENCES Country(country_code),
                                        FOREIGN KEY (cancellation_policy) REFERENCES
Policy(cancellation_policy) );
```

New ER diagram:



Assumptions

General changes:

We decided to split the 'accomodation' entity into different smaller entities each with one or two attributes. Since most of these attributes can be reused for different listings, we need to generate all possible combinations of these entities, and by splitting them we decrease drastically the number of generated combinations. We did not split policy because it has up to 16 combinations, which is reasonable.
Other change we made was turning the 'Pricing' entity into a full one, as opposed to a weak entity. The entity can be reused by another listing which has the exact same values, so it should not be deleted when a listing is deleted.

A small change was the one-to-one relation of liting with Description. This will be unique for that listing so this constraint had to be added.

Another small change was made to the relation of Calendar with Listing. After checking the data we decided that it was not true that there should be an 'at least one constraint' here, so it was removed.

The 'host_neighbourhood' attribute in Host was removed. Insteaad we decided to add a direct relation of Host with Neighborhood. This makes a lot of sense from a practical point of view (there was no need or advantage to keep the neighbourhood as an attribute).

Finally, we changed the Review Entity. We had misunderstood the significance of the attributes such as 'review_scores_rating', 'review_score_accuracy' etc. as multiple ratings given by in each review. What they were actually supposed to do was represent an aggregation of the scores received by all the reviews for that listing. That's why we decided to put all of these attributes as a weak entity named 'Score' which has a one-to-one relation with Listing. The two atributes ('comments' and 'review_date') which were actually given by each review to the listing, are now set as attributes of the 'reviewed' relation.

Data Loading

Bedding had NAN values, which we replaced with 0

Description: every NAN was replaced either eith an empty string or wit -1 in the case of square feet

The description_id is an incrementing number starting from 0

Reviewer: The data contained two users with the same reviewer_id but different reviewer_name so we decided to drop one of them (casa nuna) as it is an anomally.

Pricing: Most of the listings contained values for the daily price but not the weekly and the monthly. So, to fill the missing entries we multiplied the daily price by 7 and 30 accordingly. For the security deposit and the cleaning fee we decided to put 0 in the missing entries.

Policy: With regards to the old ER table we removed the policy_id and replaced it with cancellation_policy. The other boolean attributes were transformed to relationship attributes.

Score: There were listings that had no reviews so all their entries where NaN. We replaced these values with -1.

Provides had some duplicate lines, meaning that some listings mentioned some amenities more than once. We dropped them.

While observing the data, we realized that users inserted Madrid, Barcelona and Berlin with many different ways. We decided to remove any confusion from future queries by replacing all that by just the proper names of the cities.

In the Listing entity we added the host_name because it drastically fascilitated our queries.

Basic setting up

```
In [1]: import pandas as pd
        import numpy as np
        from sqlalchemy import create_engine
        import sqlalchemy
        import datetime
```

```
In [2]: listings = pd.read_csv("barcelona_listings.csv")
```

```
In [3]: reviews = pd.read_csv("barcelona_reviews.csv")
```

```
In [4]: calendar = pd.read_csv("barcelona_calendar.csv")
```

```
In [5]: engine = create_engine("sqlite:///project.db")
```

```
In [6]: def null_count(x):
            return listings[x].isna().sum()# / len(listings[x])

        def unique_count(x):
            return listings.nunique()# / listings.shape[0] * 100

        #for i,c in enumerate(listings.columns):
         #   print(c, unique_count(c)[i])
```

Import accommodates

```
In [7]: df = pd.DataFrame(listings.accommodates.unique(), columns=["accommodates"])
```

```
In [9]: df.to_sql("Accommodation", engine, if_exists="append", index=False, dtype={"ac
        commodates" : sqlalchemy.INT})
```

```
In [ ]: engine.execute("select count(*) from Accommodation ").fetchall()
```

Import amenities

```
In [7]: amenities = pd.DataFrame(listings.amenities.unique(), columns=["amenities"])
```

```
In [8]: temp = []
        for index, row in amenities.iterrows():
            temp.append(row["amenities"].split(','))
```

In [9]:
```
flat_list = []
for sublist in temp:
    for item in sublist:
        flat_list.append(item)
```

In [10]:
```
def remove_chars(item):
    for c in item:
        if c == "{" or c == "}" or c == "'" or c == '"' or c == "[" or c ==
"]":
            item = item.replace(c, "")
        item = item.strip().lower()
    return item
```

In [21]:
```
new_list = []
for item in flat_list:
    new_list.append('"' + remove_chars(item) + '"')
```

In [12]:
```
new_list = list(set(new_list))
```

In [13]:
```
new_list.remove('""')
```

In [14]:
```
amenities = pd.DataFrame(new_list, columns=["amenities"])
```

In [93]:
```
amenities.to_sql("Amenities", engine, if_exists="append", index=False, dtype={
"amenities" : sqlalchemy.CHAR(32)})
```

In [61]:
```
bathrooms = pd.DataFrame(listings.bathrooms.unique(), columns=["bathrooms"])
```

In [62]:
```
bathrooms = bathrooms.dropna()
```

In [63]:
```
bathrooms.to_sql("Bathrooms", engine, if_exists="append", index=False, dtype={
"bathrooms" : sqlalchemy.FLOAT})
```

Import Description

In [10]:
```
description = pd.DataFrame({"description_id": np.arange(len(listings.descripti
on)), "summary": listings.summary, "space": listings.space, "description": lis
tings.description, "neighborhood_overview": listings.neighborhood_overview,"no
tes":listings.notes,"transit":listings.transit,"access":listings.access,"pictu
re_url":listings.picture_url,"square_feet": listings.square_feet, "id": listin
gs.id})
```

In [11]:
```
description["space"].fillna("",inplace=True)
description["neighborhood_overview"].fillna("",inplace=True)
description["description"].fillna("",inplace=True)
description["summary"].fillna("",inplace=True)
description["notes"].fillna("",inplace=True)
description["transit"].fillna("",inplace=True)
description["access"].fillna("",inplace=True)
description["picture_url"].fillna("",inplace=True)
description["square_feet"].fillna(-1,inplace=True)
```

In [14]:
```
description.to_sql("Described_Description", engine, if_exists="append", index=
False, dtype={"description_id": sqlalchemy.INT, "summary": sqlalchemy.VARCHAR(
1024), "space": sqlalchemy.VARCHAR(1024), "description": sqlalchemy.VARCHAR(10
24), "neighborhood_overview": sqlalchemy.VARCHAR(1024),"notes":sqlalchemy.VARC
HAR(1024),"transit": sqlalchemy.VARCHAR(1024),"access": sqlalchemy.VARCHAR(102
4),"picture_url":sqlalchemy.CHAR(128),"square_feet": sqlalchemy.FLOAT, "id": s
qlalchemy.INT})
```

Import bedding

In [19]:
```
bedding = pd.DataFrame({"beds": listings.beds, "bed_type": listings.bed_type})
```

In [21]:
```
bedding = bedding.fillna(0).drop_duplicates()
```

In [22]:
```
bedding.to_sql("Bedding", engine, if_exists="append", index=False, dtype={"bed
s" : sqlalchemy.FLOAT, "bed_type" : sqlalchemy.CHAR(32)})
```

Import Bedrooms

In [4]:
```
bedrooms = pd.DataFrame({"bedrooms": listings.bedrooms})
```

In [10]:
```
bedrooms = bedrooms.dropna().drop_duplicates()
```

In [13]:
```
bedrooms.to_sql("Bedrooms", engine, if_exists="append", index=False, dtype={"b
edrooms" : sqlalchemy.INT})
```

Import Room

In [14]:
```
room = pd.DataFrame({"room_type": listings.room_type})
```

In [17]:
```
room = room.drop_duplicates()
```

In [18]:
```
room.to_sql("Room", engine, if_exists="append", index=False, dtype={"room_typ
e" : sqlalchemy.CHAR(32)})
```

Import Property

In [19]:
```
prop = pd.DataFrame({"property_type": listings.property_type})
```

In [23]:
```
prop = prop.drop_duplicates()
```

In [24]:
```
prop.to_sql("Property", engine, if_exists="append", index=False, dtype={"prope
ry_type" : sqlalchemy.CHAR(32)})
```

Import Reviewer

In [29]:
```
reviewer = pd.DataFrame({"reviewer_id": reviews.reviewer_id, "reviewer_name" :
 reviews.reviewer_name})
```

In [50]:
```
reviewer = reviewer.drop_duplicates().drop(174201)
```

In [51]:
```
reviewer.to_sql("Reviewer", engine, if_exists="append", index=False, dtype={"r
eviewer_id" : sqlalchemy.INT, "reviewer_name" : sqlalchemy.CHAR(32)})
```

Import Reviewed TODO

In [94]:
```
reviewed = pd.DataFrame({"id": reviews.listing_id, "reviewer_id" : reviews.rev
iewer_id, "comments": reviews.comments, "review_date" : reviews.date})
```

We need to change the primary key to (id, reviewer_id, date) and decide what to do with the duplicates

Import Pricing

In [118]:
```
pricing = pd.DataFrame({"price": listings.price, "weekly_price" : listings.wee
kly_price, "monthly_price": listings.monthly_price, "security_deposit" : listi
ngs.security_deposit, "cleaning_fee" : listings.cleaning_fee, "guests_include
d" : listings.guests_included, "extra_people" : listings.extra_people, "minimu
m_nights" : listings.minimum_nights, "maximum_nights" : listings.maximum_night
s, "id" : listings.id})
```

In [119]:
```
for i in pricing.index:
    pricing.at[i, "price"] = float(str(pricing.at[i, "price"])[1:].replace(","
, ""))
    if not pd.isnull(pricing.at[i, "weekly_price"]):
        pricing.at[i, "weekly_price"] = float(str(pricing.at[i, "weekly_price"
])[1:].replace(",", ""))
    if not pd.isnull(pricing.at[i, "monthly_price"]):
        pricing.at[i, "monthly_price"] = float(str(pricing.at[i, "monthly_pric
e"])[1:].replace(",", ""))
    if not pd.isnull(pricing.at[i, "security_deposit"]):
        pricing.at[i, "security_deposit"] = float(str(pricing.at[i, "security_
deposit"])[1:].replace(",", ""))
    if not pd.isnull(pricing.at[i, "cleaning_fee"]):
        pricing.at[i, "cleaning_fee"] = float(str(pricing.at[i, "cleaning_fee"
])[1:].replace(",", ""))
    if not pd.isnull(pricing.at[i, "extra_people"]):
        pricing.at[i, "extra_people"] = float(str(pricing.at[i, "extra_people"
])[1:].replace(",", ""))
```

In [120]:
```
for i in pricing.index:
    if pd.isnull(pricing.at[i, "weekly_price"]):
        pricing.at[i, "weekly_price"] = pricing.at[i, "price"] * 7
    if pd.isnull(pricing.at[i, "monthly_price"]):
        pricing.at[i, "monthly_price"] = pricing.at[i, "price"] * 30
    if pd.isnull(pricing.at[i, "security_deposit"]):
        pricing.at[i, "security_deposit"] = 0
    if pd.isnull(pricing.at[i, "cleaning_fee"]):
        pricing.at[i, "cleaning_fee"] = 0
```

In [123]:
```
pricing.to_sql("Pricing", engine, if_exists="append", index=False, dtype={"pri
ce": sqlalchemy.FLOAT, "weekly_price" : sqlalchemy.FLOAT, "monthly_price": sql
alchemy.FLOAT, "security_deposit" : sqlalchemy.FLOAT, "cleaning_fee" : sqlalch
emy.FLOAT, "guests_included" : sqlalchemy.INT, "extra_people" : sqlalchemy.FLO
AT, "minimum_nights" : sqlalchemy.INT, "maximum_nights" : sqlalchemy.INT, "id"
 : sqlalchemy.INT})
```

Import Verification

In [50]:
```
verification = pd.DataFrame(listings.host_verifications.unique(), columns=["ho
st_verifications"])
```

In [51]:
```
temp = []
for index, row in verification.iterrows():
    temp.append(row["host_verifications"].split(','))
```

In [52]:
```
flat_list = []
for sublist in temp:
    for item in sublist:
        flat_list.append(item)
```

In [55]:
```
new_list = []
for item in flat_list:
    new_list.append('"' + remove_chars(item) + '"')
```

In [56]:
```
new_list = list(set(new_list))
```

In [57]:
```
new_list.remove('""')
```

In [59]:
```
verification = pd.DataFrame({"host_verifications" : new_list})
```

In [61]:
```
verification.to_sql("Verification", engine, if_exists="append", index=False, d
type={"host_verifications" : sqlalchemy.CHAR(32)})
```

Import verified_by TODO

Import Calendar

In [40]:
```
cal = pd.DataFrame({"date" : calendar.date.unique()})
```

In [41]:
```
for i in cal.index:
    cal.at[i, "date"] = datetime.datetime.strptime(str(cal.at[i, "date"]), "%Y
-%m-%d").date()
```

In [42]:
```
cal.to_sql("Calendar", engine, if_exists="append", index=False, dtype={"date"
: sqlalchemy.DATE})
```

Import Available_at

In [43]:
```
avail = pd.DataFrame({"id" : calendar.listing_id, "date" : calendar.date, "pri
ce" : calendar.price, "available" : calendar.available})
```

In [48]:
```
for i in calendar.index:
    avail.at[i, "date"] = datetime.datetime.strptime(str(calendar.at[i, "date"
]), "%Y-%m-%d").date()
```

In [44]:
```
for i in avail.index:
    if not pd.isna(avail.at[i, "price"]):
        avail.at[i, "price"] = float(str(avail.at[i, "price"]).replace(",","")
.replace("$", ""))
```

In [45]:
```
avail = avail.fillna(-1)
```

In [49]:
```
avail.to_sql("Available_at", engine, if_exists="append", index=False, dtype={
"id" : sqlalchemy.INT, "date" : sqlalchemy.DATE, "price" : sqlalchemy.INT, "av
ailable" : sqlalchemy.CHAR(1)})
```

Import Policy

In [28]:
```
reduced = pd.DataFrame({"cancellation_policy" : listings["cancellation_policy"
]})
```

In [34]:
```
reduced = reduced.drop_duplicates()
```

In [35]:
```
reduced.to_sql("Policy", engine, if_exists="append", index=False, dtype={"canc
ellation_policy" : sqlalchemy.CHAR(32)})
```

Import Score

In [70]:
```
score = pd.DataFrame({"review_scores_rating" : listings.review_scores_rating,
"review_scores_accuracy" : listings.review_scores_accuracy, "review_scores_cle
anliness" : listings.review_scores_cleanliness, "review_scores_checkin" : list
ings.review_scores_checkin, "review_scores_communication" : listings.review_sc
ores_communication, "review_scores_location" : listings.review_scores_location
, "review_scores_value" : listings.review_scores_value, "id" : listings.id})
```

In [71]:
```
score = score.fillna(-1)
```

In [72]:
```
score.to_sql("Score", engine, if_exists="append", index=False, dtype={"review_
scores_rating" : sqlalchemy.INT, "review_scores_accuracy" : sqlalchemy.INT, "r
eview_scores_cleanliness" : sqlalchemy.INT, "review_scores_checkin" : sqlalche
my.INT, "review_scores_communication" : sqlalchemy.INT, "review_scores_locatio
n" : sqlalchemy.INT, "review_scores_value" : sqlalchemy.INT, "id" : sqlalchemy
.INT})
```

Import Host

In [79]:
```
host = pd.DataFrame({"host_id" : listings.host_id, "host_since" : listings.hos
t_since, "host_about" : listings.host_about, "host_response_time" : listings.h
ost_response_time, "host_response_rate" : listings.host_response_rate, "host_n
eighbourhood" : listings.host_neighbourhood, "host_url" : listings.host_url,
"host_name" : listings.host_name, "host_thumbnail_url" : listings.host_thumbna
il_url, "host_picture_url" : listings.host_picture_url})
```

In [80]:
```
host = host.fillna("Unknown")
```

In [86]:
```
for i in host.index:
    host.at[i, "host_since"] = datetime.datetime.strptime(str(host.at[i, "host
_since"]), "%Y-%m-%d").date()
```

In [90]:
```
host = host.drop_duplicates()
```

In [91]:
```
host.to_sql("Host", engine, if_exists="append", index=False, dtype={"host_id"
: sqlalchemy.CHAR(32), "host_since" : sqlalchemy.DATE, "host_about" : sqlalche
my.VARCHAR(1024), "host_response_time" : sqlalchemy.CHAR(32), "host_response_r
ate" : sqlalchemy.CHAR(32), "host_neighbourhood" : sqlalchemy.CHAR(32), "host_
url" : sqlalchemy.CHAR(32), "host_name" : sqlalchemy.CHAR(32), "host_thumbnail
_url" : sqlalchemy.CHAR(32), "host_picture_url" : sqlalchemy.CHAR(32)})
```

We need to add host_neighbourhood, city and country, infer it from the general dataset

In [ ]:
```

```

Import Provide

In [61]:
```
provide = listings[["id", "amenities"]]
```

In [62]:
```
temp2 = []
for index, row in provide.iterrows():
    temp2.append(row["amenities"].split(','))
```

In [63]:
```
removed_chars_list = []
for l in temp2:
    tmp = []
    for a in l:
        tmp.append(remove_chars(a))
    removed_chars_list.append(tmp)
```

In [64]:
```
ids = []
amens = []
for i, elem in enumerate(provide["id"]):
    for j in np.arange(len(removed_chars_list[i])):
        ids.append(elem)
        amens.append(removed_chars_list[i][j])
```

In [67]:
```
provides = pd.DataFrame({"id" : ids, "amenities" : amens})
```

In [71]:
```
provides = provides.drop_duplicates()
```

In [72]:
```
provides.to_sql("Provides", engine, if_exists="append", index=False, dtype={"i
d" : sqlalchemy.INT, "amenities" : sqlalchemy.CHAR(32)})
```

Country

In [138]:
```
country = ["Spain", "Germany"]
c_code = ["ES", "DE"]
```

In [139]:
```
da_country = pd.DataFrame({"country" : country, "country_code" : c_code})
```

In [143]:
```
da_country.to_sql("Country", engine, if_exists="append", index=False, dtype={
"country_code" : sqlalchemy.CHAR(2), "country" : sqlalchemy.CHAR(32)})
```

City

In [133]:
```
city= ["Barcelona","Madrid","Berlin"]
```

In [134]:
```
country_code = ["ES", "ES", "DE"]
```

In [135]:
```
da_city = pd.DataFrame({"city" : city, "country_code" : country_code})
```

In [ ]:

In [137]:
```
da_city.to_sql("City", engine, if_exists="append", index=False, dtype={"city"
: sqlalchemy.CHAR(32), "country_code" : sqlalchemy.CHAR(2)})
```

In [ ]:

Neighbourhood

In [98]:
```
nei = listings[["neighbourhood", "country_code", "city"]]
```

In [109]:
```
nei.drop_duplicates(inplace=True)
```

```
/anaconda3/lib/python3.6/site-packages/ipykernel_launcher.py:1: SettingWithCo
pyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/st
able/indexing.html#indexing-view-versus-copy
  """Entry point for launching an IPython kernel.
```

In [105]:
```
for i in nei.index:
    nei.at[i, "city"] = "Barcelona"
```

In [130]:
```
nei.to_sql("Neighbourhood", engine, if_exists="append", index=False, dtype={"N
eighbourhood" : sqlalchemy.CHAR(32), "country_code" : sqlalchemy.CHAR(2), "cit
y" : sqlalchemy.CHAR(32)})
```

Import Listing

In [144]:
```
da_listing = listings[["id", "listing_url", "name", "accommodates", "cancellat
ion_policy", "host_id", "host_name", "neighbourhood", "city", "country_code",
"latitude", "longitude", "property_type", "room_type", "bathrooms", "bedrooms"
, "beds", "bed_type", "interaction", "house_rules", "is_business_travel_ready"
, "require_guest_profile_picture", "require_guest_phone_verification"]]
```

In [146]:
```
da_listing.name = da_listing.name.fillna("")
da_listing.interaction = da_listing.interaction.fillna("")
da_listing.house_rules = da_listing.house_rules.fillna("")

da_listing.city = da_listing.city.fillna("Barcelona")
da_listing.bathrooms = da_listing.bathrooms.fillna(0)
da_listing.bedrooms = da_listing.bedrooms.fillna(0)
da_listing.beds = da_listing.beds.fillna(0)
```

```
/anaconda3/lib/python3.6/site-packages/pandas/core/generic.py:4405: SettingWi
thCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/st
able/indexing.html#indexing-view-versus-copy
  self[name] = value
```

In [150]:
```
da_listing.to_sql("Listing", engine, if_exists="append", index=False, dtype={
"id" : sqlalchemy.INT, "listing_url" : sqlalchemy.CHAR(32), "name" : sqlalchem
y.CHAR(32), "accommodates" : sqlalchemy.CHAR(32), "cancellation_policy" : sqla
lchemy.CHAR(32), "host_id" : sqlalchemy.INT,  "host_name" : sqlalchemy.CHAR(32
), "neighbourhood": sqlalchemy.CHAR(32), "city" : sqlalchemy.CHAR(32), "countr
y_code" : sqlalchemy.CHAR(2), "latitude" : sqlalchemy.FLOAT, "longitude" : sql
alchemy.FLOAT, "property_type": sqlalchemy.CHAR(32), "room_type" : sqlalchemy.
CHAR(32), "bathrooms" : sqlalchemy.FLOAT, "bedrooms" : sqlalchemy.INT, "beds"
: sqlalchemy.INT, "bed_type" : sqlalchemy.CHAR(32), "interaction" : sqlalchemy
.VARCHAR(1024), "house_rules" : sqlalchemy.VARCHAR(1024), "is_business_travel_
ready" : sqlalchemy.CHAR(1), "require_guest_profile_picture" : sqlalchemy.CHAR
(1), "require_guest_phone_verification" : sqlalchemy.CHAR(1)})
```

In [ ]:

Query Implementation

```
SELECT AVG(P.price)
FROM Listing L, Pricing P
WHERE L.id=P.id AND L.beds=8 ;

SELECT AVG(S.review_scores_cleanliness)
FROM Listing L, Amenities A, SCORE S, PROVIDES P
WHERE L.id=S.id AND P.id=L.id AND P.amenities='tv' ;

SELECT DISTINCT L.host_id FROM Listing L, AVAILABLE_AT AV WHERE AV.id=L.id AND AV.available='t' AND
AV.date <= '2019-09-31' AND AV.date >= '2019-03-00' ;

SELECT COUNT(*)
FROM Listing L1
WHERE EXISTS (SELECT L2.host_name
                                        FROM Listing L2
                                        WHERE L1.host_id < L2.host_id AND
                                                      L1.host_name = L2.host_name) ;


SELECT DISTINCT AV.date
FROM Listing L, AVAILABLE_AT AV
WHERE L.host_name='Viajes Eco' AND AV.id= L.id AND AV.available='t' ;

SELECT DISTINCT L.host_id, L.host_name
FROM Listing L
GROUP BY (L.host_id)
HAVING COUNT (*) = 1 ;

SELECT AVG(L1.price) - AVG(L2.price)
FROM Listing L1, Listing L2
WHERE L1.id IN (SELECT L3.id
                              FROM Listing L3, PROVIDE P1
                              WHERE L3.id=P1.id AND P1.amenities='wifi')
   AND L2.id NOT IN (SELECT L4.id
                                             FROM Listing L4, PROVIDE P2
                                             WHERE L4.id=P2.id AND P2.amenities='wifi') ;

SELECT AVG(L1.price) - AVG(L2.price)
FROM Listing L1, Listing L2
WHERE L1.neighborhood='Berlin' AND L2.neighborhood='Madrid'
        AND L1.beds=L2.beds AND L1.beds=8 ;

SELECT TOP (10)
L.host_id,L.host_name
FROM Listing L
WHERE L.country='Spain'
ORDER BY (SELECT COUNT(*)
               FROM Listing L1
    WHERE L.host_id = L1.host_id AND L.id < L1.id) DESC ;

SELECT TOP (10)
L.id,L.name
```

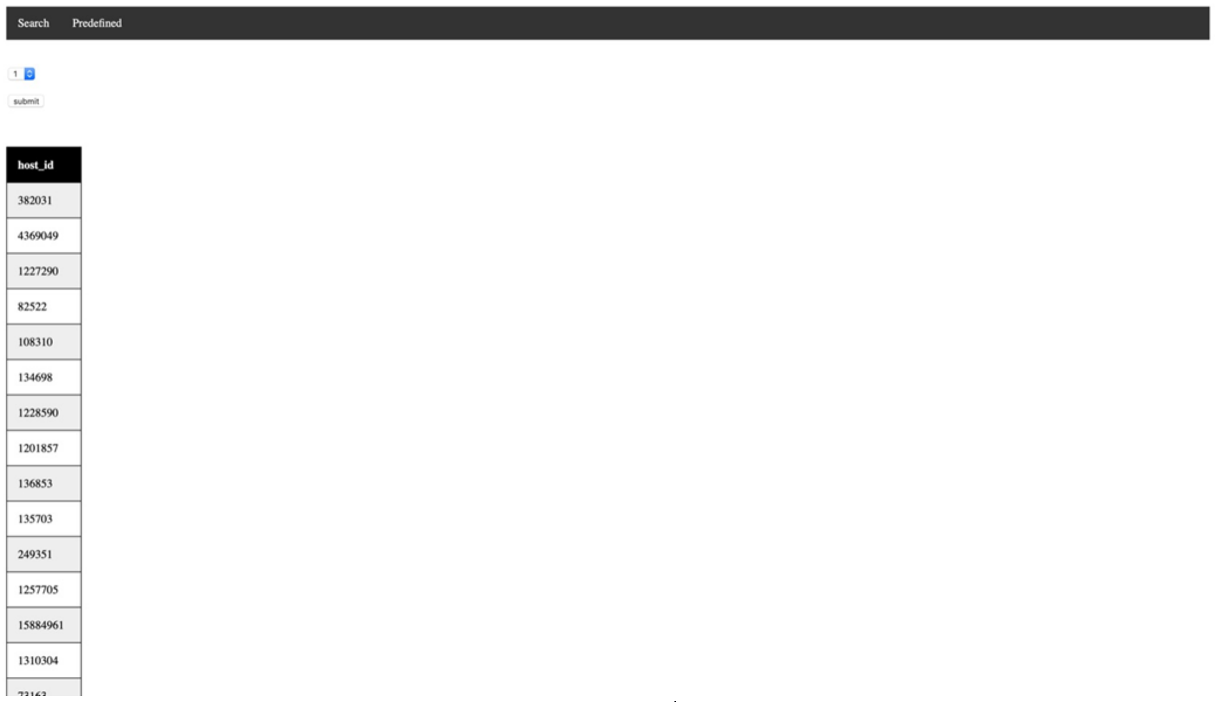
```
FROM Listing L, Score S
WHERE L.neighborhood='Barcelona', S.id=L.id
ORDER BY (S.review_scores_cleanliness) DESC ;
```

Interface

Design logic Description

The UI will be fairly simple, it has a search tab (where you can search by keywords) and another tab where you can run the predefined queries.
The way we implemented it was using Flask, a python library which runs a local server on your machine, from which you can access html files. The data

Screenshots



host_id
382031
4369049
1227290
82522
108310
134698
1228590
1201857
136853
135703
249351
1257705
15884961
1310304
[illegible]

This is a screenshot of what it looks like if you run the 3rd predefined query. The user can select a query from the drop-down menu and then run it. In the other tab the user will be able to search by a keyword, but we did not have time to finish this before the deadline this time. (This is how we intend to search: https://hashnode.com/post/sqlite-fts5-full-text-searching-cjmklqx50000d6is2lkuspn09)

General Comments

We decided to use Sqlite for our database. It was easier and faster to do it this way than accessing it remotely every single time. We used pandas for the data cleaning and sqlalchemy for when inserting into our sqlite database. The way we implemented the UI was using Flask, a python library which runs a local server on your machine, from which you can access html files.

George- Primarily worked on the data cleaning process
Gerald- Primarily worked on the UI
Ridha- Primarily worked on the predefined queries
All three worked together on the decision making process that we went through when making changes to our ER diagram and Tables, and during the process of deciding how to clean the data.

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/

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Deliverable 3

Assumptions

<In this section write down the assumptions you made about the data. Write a sentence for each assumption you made>

Query Implementation

<For each query>

Query a:

Description of logic:

<What does the query do and how do I decide to solve it> *SQL statement*

<The SQL statement>

Query Analysis

Selected Queries (and why)

Query 1

<Initial Running time:
Optimized Running time:
Explain the improvement:
Initial plan
Improved plan>

Query 2

<Initial Running time:
Optimized Running time:
Explain the improvement:
Initial plan
Improved plan>

Query 3

<Initial Running time:
Optimized Running time:
Explain the improvement:
Initial plan
Improved 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/

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

Interface

Design logic Description

<Describe the general logic of your design as well as the technology you decided to use>

Screenshots

<Provide some initial screen shots of your interface>

General Comments

<In this section write general comments about your deliverable (comments and work allocation between team members>