**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**

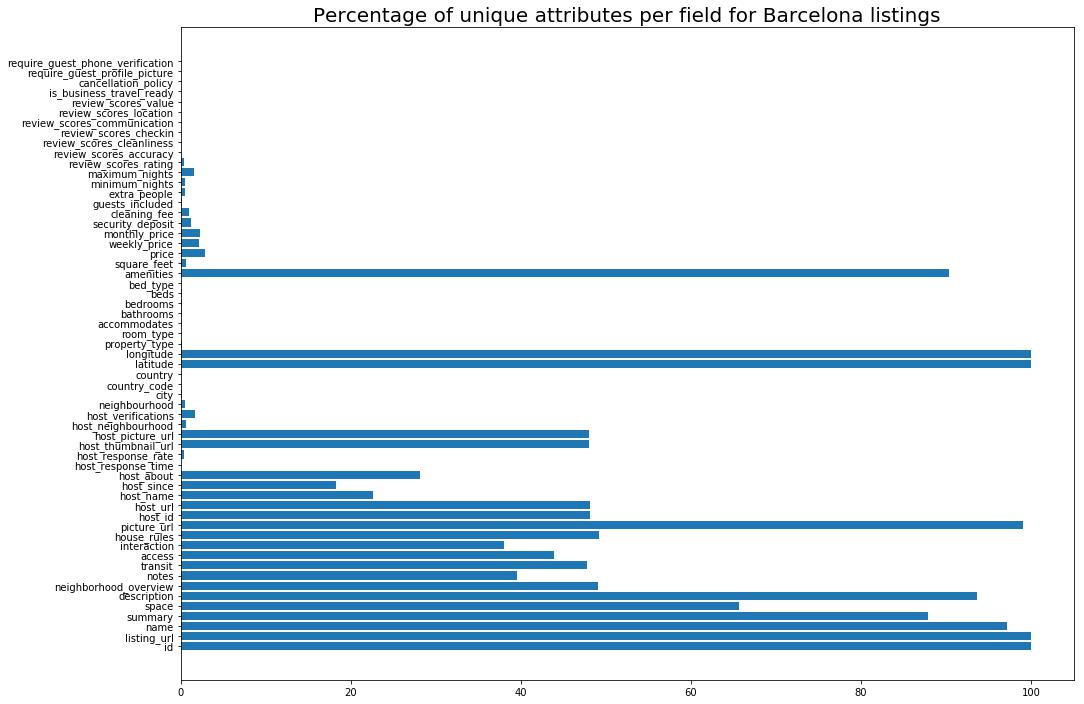
School of Computer and Communication Sciences

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**

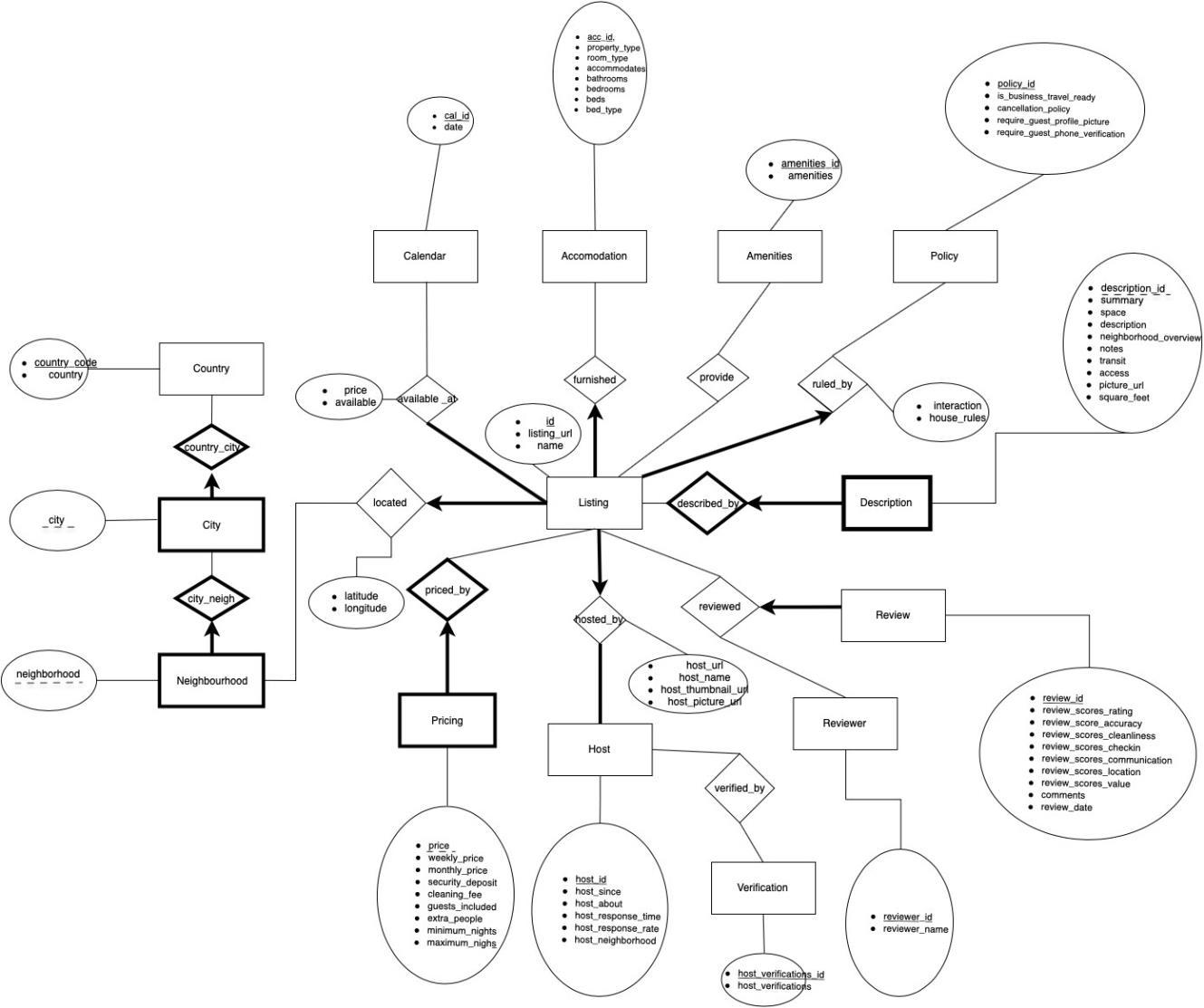
School of Computer and Communication Sciences

Ecole Polytechnique Fédérale de Lausanne

Building BC, Station 14

CH-1015 Lausanne

URL: <http://dias.epfl.ch/>



Description

**Calendar**: there is a ‘one to many’ relation with listing because one listing will have at least one date ofavailability. The ‘price’ and ‘available’ attributes were set as relational attributes, because they were different for different listings (as they should be).

**Accommodation**: This entity describes the interior of the listing’s apartment. All of these attributes will changefrom 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.

**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/>

**Amenities**: The data contained here would make sense to be included in the accommodation entity, but sincelistings 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. Wemade 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 Citycan 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),

**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/>

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),

**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/>

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),

**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/>

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),

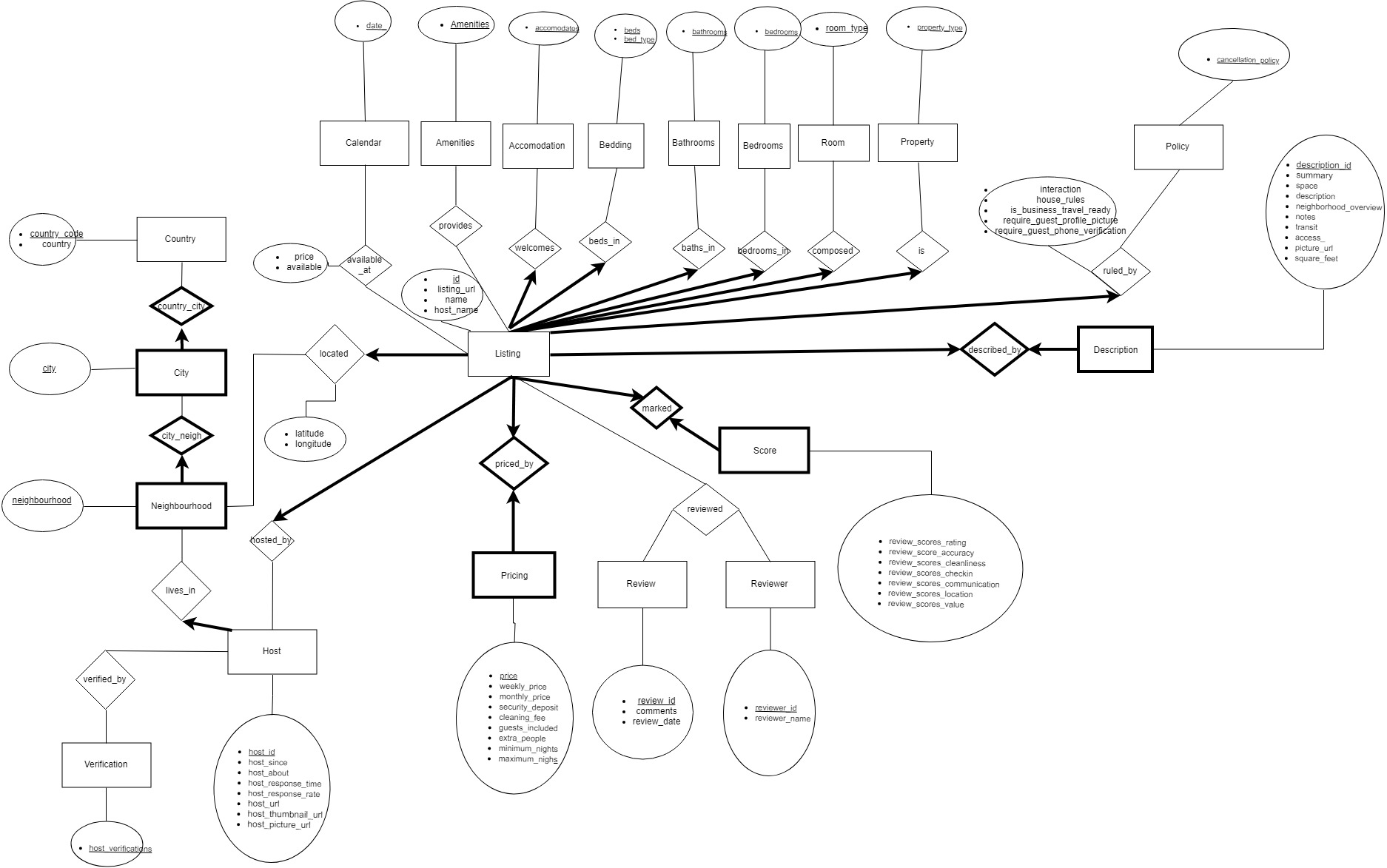
FOREIGN KEY (country\_code,city,neighbourhood) REFERENCES Neighbourhood(country\_code,city,neighbourhood),

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.

***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

******

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/>

**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/>

**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>