Databases Project – Spring 2019

Team No: 43

Names: Stanislas Michel Jouven, Valentin Quelquejay-Leclere, Robin Adrien Zbinden

Contents

Contents 1

Deliverable 1 2

Assumptions 2

Entity Relationship Schema 2

Schema 2

Description 2

Relational Schema 2

ER schema to Relational schema 2

DDL 3

General Comments 3

Deliverable 2 4

Assumptions 4

Data Loading 4

Query Implementation 4

Query a: 4

Description of logic: 4

SQL statement 4

Interface 4

Design logic Description 4

Screenshots 4

General Comments 4

Deliverable 3 5

Assumptions 5

Query Implementation 5

Query a: 5

Description of logic: 5

SQL statement 5

Query Analysis 5

Selected Queries (and why) 5

Query 1 5

Query 2 5

Query 3 5

Interface 6

Design logic Description 6

Screenshots 6

General Comments 6

# Deliverable 1

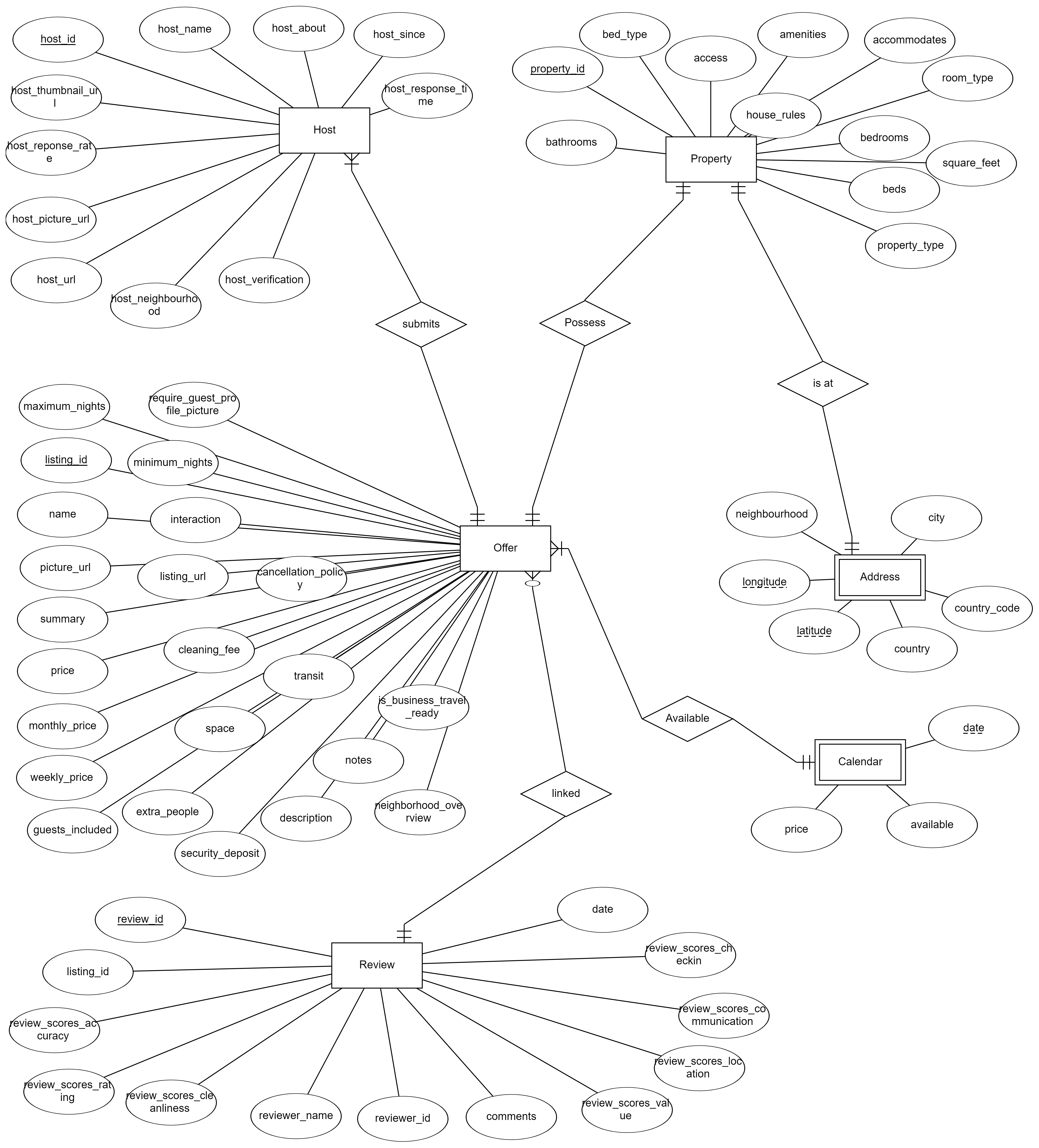
## Assumptions

We have done the following assumptions:

* We assumed that all the ids are uniquely defined.
* First, we have assumed that the reviewer\_id is different from the host\_id even if it’s the same user.

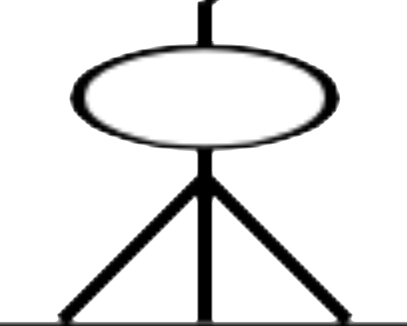
## Entity Relationship Schema

### Schema



Legend:

* : At least one (mandatory)
* : Exactly one (mandatory)

 : Many, optional

### 

### Description

First, we have separated the offer (the informations that are used to sell on the website) of the property (the informations that are intrinsic to the physical identity). In order to be able to filter offers by accomodation specifications, we thought it would make sense to have two different entities. For example, by doing this way, it would be easier to search for accomodations with a particular number of beds or area. Of course, each offer has only one property and a property can have only one offer.

We have also made the choice to consider a weak entity for the address of the property because an address can’t exist by itself. Furthermore, it allows to delete the address whenever the property is deleted.

We grouped all the information about the host of a listing into a single entity. Indeed, a host can have multiple offers, and therefore we have a one to many relationship between host and offer entity.

The main point for which we decided to use a weak entity for calendar is that it cannot be uniquely determined by his attributes. Moreover, if we choose to delete an offer, it also make sense to delete the associated calendar.

We have thought about creating a user entity that would have combined the reviewer and the host. However, regarding of the way the data is presented (reviewer id and host id), we believed it was better to keep them separated. In consequence, we created a review entity that groups all the informations about a review.

## Relational Schema

### ER schema to Relational schema

We have translated the different relations as usual. We have grouped the “offer” entity and the “possesses” relationship set into one table. It’s the same thing for the “is at” relationship and the address entity. Consequently, we were not able to capture the fact that a property has mandatory participation in the offer and the address relationship set.

We have also grouped calendar and available table because of the one-to-many relationship set. It’s the same thing as before: if we can capture the fact that a calendar needs to have a reference to an offer (weak entity), we are not able to capture the fact that an offer has mandatory participation in “available” relationship set.

Furthermore, because address is a weak entity, we have added the keywords ON DELETE CASCADE after the reference to listing\_id key in order to make sure that the address is deleted if the offer is deleted.

Finally, we have also grouped the “review” entity with the “linked” relationship set. However this time, because of the one-to-many relation with optional participation, we are able to capture all the constraints by grouping these 2 tables together. A review has a foreign key on the offer entity which ensures its participation in the relation.

For now, we have inferred the types of each value based on its name and we will still have to confirm these types next sprint when parsing the data and maybe define some limits for fields such as ids.

The NOT NULL keyword was also added when needed eg. for all the different primary and foreign key fields.

### 

### DDL

CREATE TABLE Host

(

host\_reponse\_rate CHAR,

host\_thumbnail\_url CHAR,

host\_url CHAR NOT NULL,

host\_id INT NOT NULL,

host\_response\_time INT,

host\_name CHAR,

host\_verification CHAR,

host\_about CHAR,

host\_neighbourhood CHAR,

host\_since DATE,

PRIMARY KEY (host\_id)

);

CREATE TABLE Address

(

city CHAR,

neighbourhood CHAR,

longitude INT NOT NULL,

latitude INT NOT NULL,

country CHAR,

country\_code INT,

property\_id INT NOT NULL,

PRIMARY KEY (longitude, latitude, property\_id),

FOREIGN KEY (property\_id) REFERENCES Property(property\_id),

ON DELETE CASCADE

);

CREATE TABLE Calendar

(

date DATE,

available CHAR,

price INT,

listing\_id INT NOT NULL,

PRIMARY KEY (date, listing\_id)

FOREIGN KEY (listing\_id) REFERENCES Offer(listing\_id),

ON DELETE CASCADE

);

CREATE TABLE Review

(

scores\_accuracy INT,

scores\_rating INT,

review\_id INT NOT NULL,

comments CHAR,

scores\_cleanliness INT,

reviewer\_name CHAR,

scores\_value INT,

reviewer\_id INT NOT NULL,

scores\_location INT,

dates DATE,

scores\_communication INT,

scores\_checkin INT,

listing\_id INT NOT NULL,

PRIMARY KEY (review\_id),

FOREIGN KEY (listing\_id) REFERENCES Offer(listing\_id)

);

CREATE TABLE Property

(

property\_id INT NOT NULL,

bed\_type CHAR,

access CHAR,

amenities CHAR,

house\_rules CHAR,

property\_type CHAR,

square\_feet INT,

accommodates CHAR,

room\_types CHAR,

bedrooms INT,

beds INT,

bathrooms INT,

PRIMARY KEY (property\_id),

);

CREATE TABLE Offer

(

listing\_url CHAR,

interaction CHAR,

transit CHAR,

require\_guest\_profile\_picture CHAR,

cancelation\_policy CHAR,

notes CHAR,

is\_business\_travel\_ready CHAR,

name CHAR NOT NULL,

picture\_url CHAR,

maximum\_nights INT,

neighbourhood\_overview CHAR,

minimum\_nights INT,

summary CHAR,

price INT,

description CHAR,

extra\_people INT,

listing\_id INT NOT NULL,

guests\_included INT,

weekly\_price INT,

cleaning\_fees INT,

montly\_price INT,

space INT,

security\_deposit INT,

host\_id INT NOT NULL,

property\_id INT NOT NULL,

PRIMARY KEY (listing\_id),

FOREIGN KEY (host\_id) REFERENCES Host(host\_id),

FOREIGN KEY (property\_id) REFERENCES Property(property\_id)

);

## General Comments

We had a lot of discussions about how to separate the different attributes between the entities when doing the entity relationship schema. We have made some coherent choices, but we’re still not sure that they are the best and the most efficient ones.

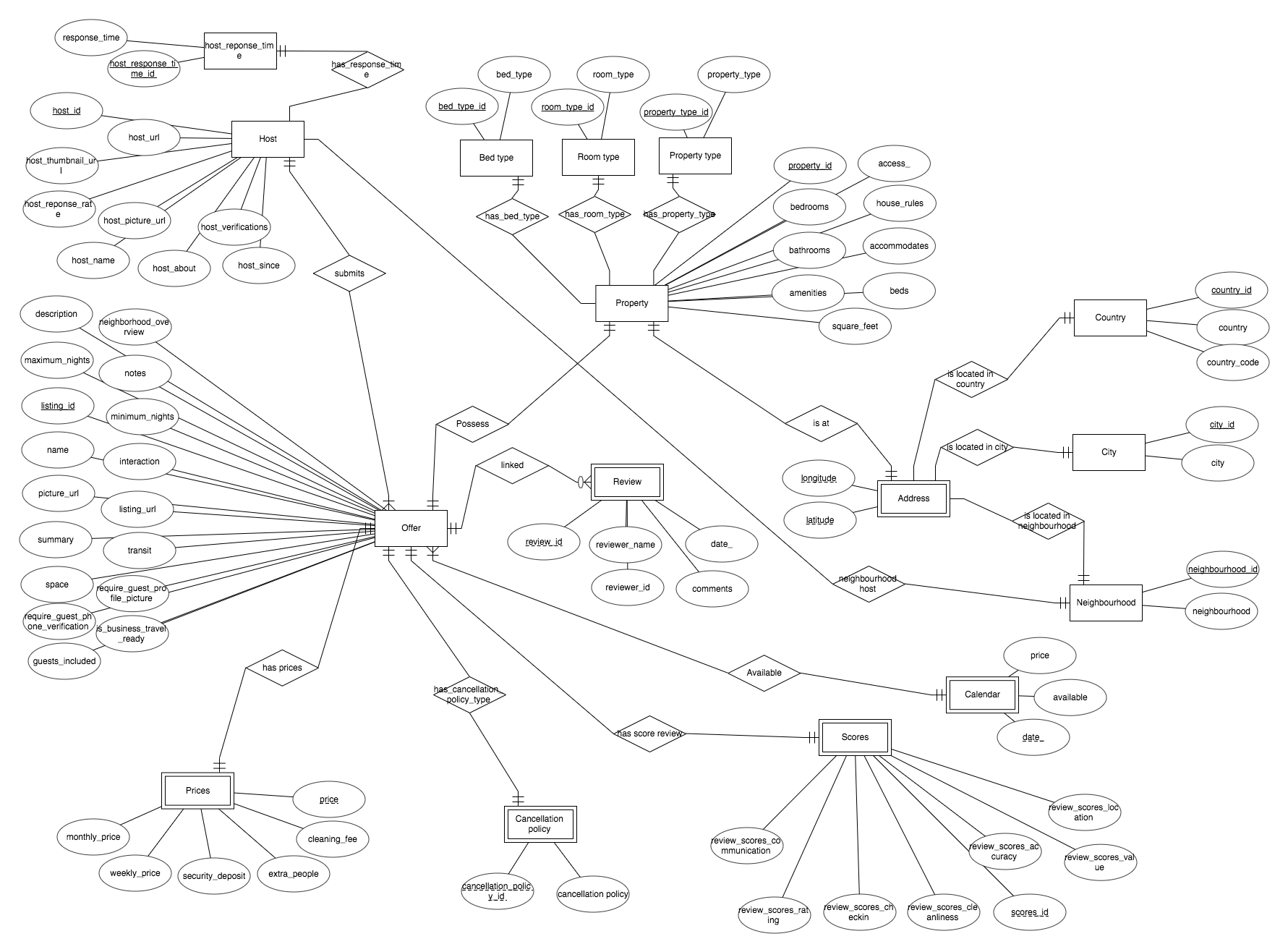
Concerning the work allocation between team members, we always work together and are aware of all the different part of this project.

# Deliverable 2

## Changes after the deliverable 1:

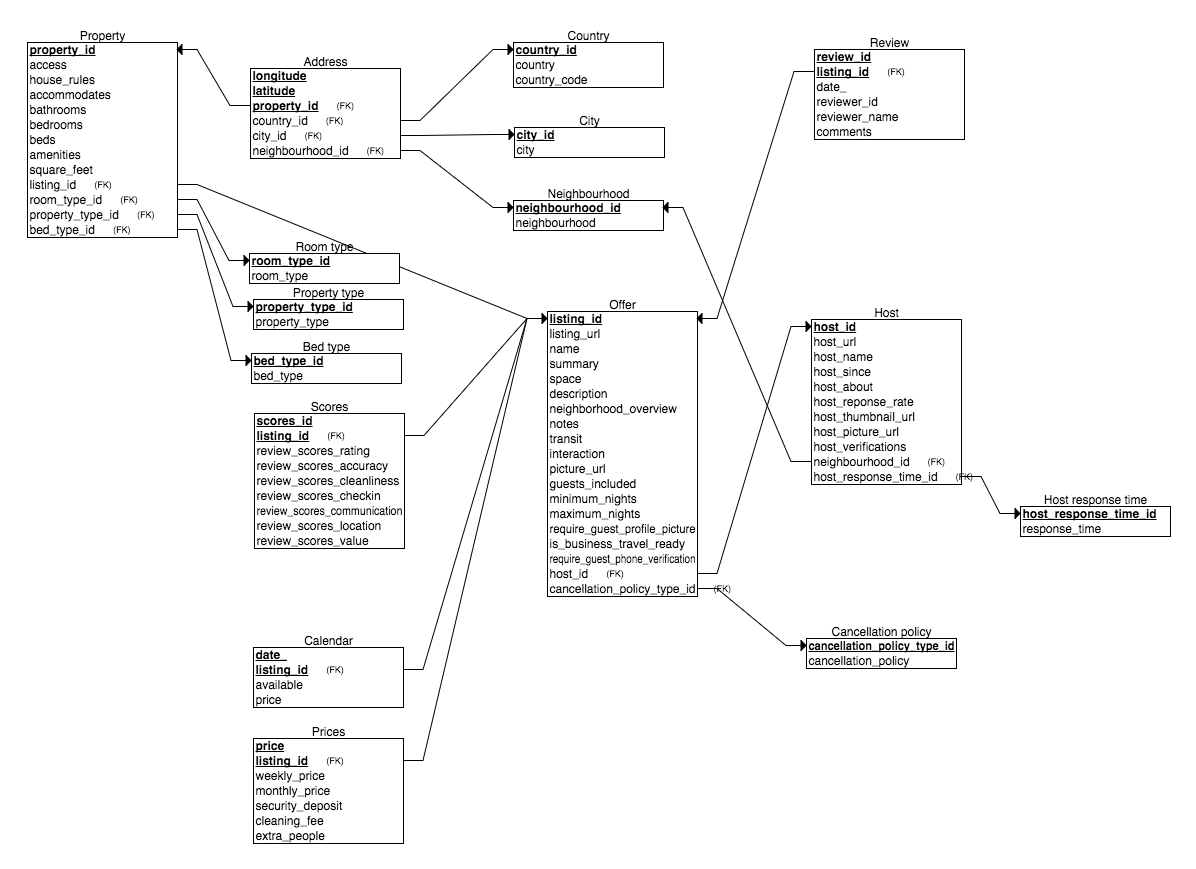
First, we forgot to say that we assumed the id of a listing from the listings files was the same than the listing\_id from the reviews files. Consequently, we deleted the attribute listing\_id in the review.

After the feedback from the TA, we decided to modify our Entity Relationship Schema as follows:



* We broke down several entities into more entities for query optimization reasons (ex: Requirements).
* To reduce data duplication, we also created new separated entities which have a predefined set of categorical values (ex: Bed type).
* To make more sense of the semantic of the data, we chose to create a hierarchy between the entities Country, City and Neigbourhood. In this way, we can easily filter the properties by their corresponding city or their corresponding country.
* Finally, we also grouped the two attributes host\_neighbourhood and neighbourhood into a single entity in order to centralize the neighbourhood info, and to make it easier to query for certain neighbourhoods without the need to scan whole tables.

It leads to the following changes for the DDL:



We have translated the different relations as usual. We have grouped:

* the “Host” entity with the “has\_reponse\_time” and “neighbourhood” relationships set,
* the “Property” entity with the “has\_bed\_type”, “has\_room\_type”, “has\_property\_type” relationship set,
* the “City” entity with the “lies in country” relationship,
* the “Neighbourhood” entity with the “lies in city” relationship.
* The “Address” with “is located in”, “is located in city”, “is located in neighbourhood” relationship set.

By doing this way and adding FOREIGN KEY from the grouped table to the other entity linked, every “one to many” relationship with optional participation are satisfied which is the case above.

We also grouped the “Offer” entity with the “submits” relationship and the “Review” entity with the “linked” relationship set. Consequently, we were not able to capture the fact that the host has mandatory participation in “submits” relationship set or that the “Offer” has mandatory participation in “linked” relationship set.

Furthermore, we had one to one relationships and so we grouped:

* “Prices” with “has prices”
* “Requirements with “has requirements”
* “Cancellation policy“ with “has\_cancellation:policy”
* “Scores” with “has\_score review”
* “Calendar” with “available”
* “Property with “possess”
* “Address” with “is at”

In all these cases we were able to capture one of the two “exactly one” relationship. For example between “Offer” and “Requirements” entities, by grouping “Review” with the “has requirements” relationship set we are not able to capture the fact that an offer has mandatory participation in “has requirements” relationship set.

Finally, “Prices”, “Requirements”, “Cancellation policy”, “Scores”, “Calendar”, “Address”, “Review” are all Weak Entities, we therefore added the keyword ON DELETE CASCADE after the foreign to the parent in order to make sure that the weak entity is deleted when the parent entity is deleted.

**security\_deposit INT**,  
 **cleaning\_fee INT**,  
 **extra\_people INT**,  
 **listing\_id INT NOT NULL**,  
 **PRIMARY KEY** (**price**, **listing\_id**),  
 **FOREIGN KEY** (**listing\_id**) **REFERENCES** Offer(**listing\_id**) **ON DELETE CASCADE**);  
  
**CREATE TABLE** Review  
(  
 **review\_id INT NOT NULL**,  
 **date\_ DATE**,  
 **reviewer\_id INT NOT NULL**,  
 **reviewer\_name VARCHAR2**(40),  
 **comments CLOB**,  
 **listing\_id INT NOT NULL**,  
 **PRIMARY KEY** (**review\_id** **CREATE TABLE** Bed\_type  
(  
 **bed\_type\_id INT NOT NULL**,  
 **bed\_type VARCHAR2**(15) **NOT NULL**,  
 **PRIMARY KEY** (**bed\_type\_id**)  
);  
  
**CREATE TABLE** Room\_type  
(  
 **room\_type\_id INT NOT NULL**,  
 **room\_type VARCHAR2**(15) **NOT NULL**,  
 **PRIMARY KEY** (**room\_type\_id**)  
);  
  
**CREATE TABLE** Property\_type  
(  
 **property\_type\_id INT NOT NULL**,  
 **property\_type VARCHAR2**(22) **NOT NULL**,  
 **PRIMARY KEY** (**property\_type\_id**)  
);  
  
**CREATE TABLE** Cancellation\_policy  
(  
 **cancellation\_policy\_id INT NOT NULL**,  
 **cancellation\_policy VARCHAR2**(30),  
 **PRIMARY KEY** (**cancellation\_policy\_id**)  
);  
  
**CREATE TABLE** Neighbourhood  
(  
 **neighbourhood\_id INT NOT NULL**,  
 **neighbourhood VARCHAR2**(40) **NOT NULL**,  
 **PRIMARY KEY** (**neighbourhood\_id**)  
);  
  
**CREATE TABLE** Host\_response\_time  
(  
 **host\_response\_time\_id INT NOT NULL**,  
 **response\_time VARCHAR2**(20) **NOT NULL**,  
 **PRIMARY KEY** (**host\_response\_time\_id**)  
);  
  
**CREATE TABLE** Host  
(  
 **host\_id INT NOT NULL**,  
 **host\_url VARCHAR2**(45),  
 **host\_name VARCHAR2**(40),  
 **host\_since DATE**,  
 **host\_about CLOB**,  
 **host\_reponse\_rate INT**,  
 **host\_thumbnail\_url VARCHAR2**(110),  
 **host\_picture\_url VARCHAR2**(110),  
 **host\_verifications VARCHAR2**(165),  
 **neighbourhood\_id INT NOT NULL**,  
 **host\_response\_time\_id INT NOT NULL**,  
 **PRIMARY KEY** (**host\_id**),  
 **FOREIGN KEY** (**neighbourhood\_id**) **REFERENCES** Neighbourhood(**neighbourhood\_id**),  
 **FOREIGN KEY** (**host\_response\_time\_id**) **REFERENCES** Host\_response\_time(**host\_response\_time\_id**)  
);  
  
**CREATE TABLE** Offer  
(  
 **listing\_url VARCHAR2**(40),  
 **name VARCHAR2**(140),  
 **summary VARCHAR2**(1000),  
 **space VARCHAR2**(1000),  
 **listing\_id INT NOT NULL**,  
 **cancellation\_policy\_id INT NOT NULL**,  
 **description VARCHAR2**(1000),  
 **neighborhood\_overview VARCHAR2**(1000),  
 **notes VARCHAR2**(1000),  
 **transit VARCHAR2**(1000),  
 **interaction VARCHAR2**(1000),  
 **picture\_url VARCHAR2**(120),  
 **guests\_included INT**,  
 **minimum\_nights INT**,  
 **maximum\_nights INT**,  
 **host\_id INT NOT NULL**,  
 **is\_business\_travel\_ready CHAR**(1),  
 **require\_guest\_profile\_picture CHAR**(1),  
 **require\_guest\_phone\_verification CHAR**(1),  
 **PRIMARY KEY** (**listing\_id**),  
 **FOREIGN KEY** (**host\_id**) **REFERENCES** Host(**host\_id**),  
 **FOREIGN KEY** (**cancellation\_policy\_id**) **REFERENCES** Cancellation\_policy(**cancellation\_policy\_id**)  
);  
  
**CREATE TABLE** Score  
(  
 **review\_scores\_rating INT**,  
 **review\_scores\_accuracy INT**,  
 **review\_scores\_cleanliness INT**,  
 **review\_scores\_checkin INT**,  
 **review\_scores\_communication INT**,  
 **review\_scores\_location INT**,  
 **review\_scores\_value INT**,  
 **scores\_id INT NOT NULL**,  
 **listing\_id INT NOT NULL**,  
 **PRIMARY KEY** (**scores\_id**, **listing\_id**),  
 **FOREIGN KEY** (**listing\_id**) **REFERENCES** Offer(**listing\_id**) **ON DELETE CASCADE**);  
  
  
**CREATE TABLE** Property  
(  
 **access\_ VARCHAR2**(1001),  
 **house\_rules VARCHAR2**(1001),  
 **accommodates INT**,  
 **bathrooms FLOAT**,  
 **bedrooms INT**,  
 **beds INT**,  
 **amenities VARCHAR2**(1389),  
 **square\_feet INT**,  
 **property\_id INT NOT NULL**,  
 **listing\_id INT NOT NULL**,  
 **room\_type\_id INT NOT NULL**,  
 **property\_type\_id INT NOT NULL**,  
 **bed\_type\_id INT NOT NULL**,  
 **PRIMARY KEY** (**property\_id**),  
 **FOREIGN KEY** (**listing\_id**) **REFERENCES** Offer(**listing\_id**),  
 **FOREIGN KEY** (**room\_type\_id**) **REFERENCES** Room\_type(**room\_type\_id**),  
 **FOREIGN KEY** (**property\_type\_id**) **REFERENCES** Property\_type(**property\_type\_id**),  
 **FOREIGN KEY** (**bed\_type\_id**) **REFERENCES** Bed\_type(**bed\_type\_id**)  
);  
  
**CREATE TABLE** Country  
(  
 **country\_id INT NOT NULL**,  
 **country VARCHAR2**(20) **NOT NULL**,  
 **country\_code CHAR**(2) **NOT NULL**,  
 **PRIMARY KEY** (**country\_id**)  
);  
  
**CREATE TABLE** City  
(  
 **city\_id INT NOT NULL**,  
 **city VARCHAR2**(30) **NOT NULL**,  
 **PRIMARY KEY** (**city\_id**)  
);  
  
**CREATE TABLE** Address  
(  
 **longitude FLOAT NOT NULL**,  
 **latitude FLOAT NOT NULL**,  
 **property\_id INT NOT NULL**,  
 **country\_id INT NOT NULL**,  
 **city\_id INT NOT NULL**,  
 **neighbourhood\_id INT NOT NULL**,  
 **PRIMARY KEY** (**longitude**, **latitude**, **property\_id**),  
 **FOREIGN KEY** (**country\_id**) **REFERENCES** Country(**country\_id**),  
 **FOREIGN KEY** (**city\_id**) **REFERENCES** City(**city\_id**),  
 **FOREIGN KEY** (**neighbourhood\_id**) **REFERENCES** Neighbourhood(**neighbourhood\_id**),  
 **FOREIGN KEY** (**property\_id**) **REFERENCES** Property(**property\_id**) **ON DELETE CASCADE**);  
  
**CREATE TABLE** Price  
(  
 **price INT NOT NULL**,  
 **weekly\_price INT**,  
 **monthly\_price INT**,  
, **listing\_id**),  
 **FOREIGN KEY** (**listing\_id**) **REFERENCES** Offer(**listing\_id**) **ON DELETE CASCADE**);  
  
**CREATE TABLE** Calendar  
(  
 **date\_ DATE NOT NULL**,  
 **available CHAR**(1),  
 **price INT**,  
 **listing\_id INT NOT NULL**,  
 **PRIMARY KEY** (**date\_**, **listing\_id**),  
 **FOREIGN KEY** (**listing\_id**) **REFERENCES** Offer(**listing\_id**) **ON DELETE CASCADE**);

## Assumptions

## Data Loading

## Query Implementation

<For each query>

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

### Query 1:

#### Description of logic:

Find the average price for a listing with 8 bedrooms.

We select all the listing\_id that have 8 bedrooms using the IN statement, then we extract the average of the prices.

#### SQL statement

SELECT AVG(prices.price)

FROM Prices price

WHERE prices.listing\_id IN (

SELECT property.listing\_id

FROM Property property

WHERE property.bedrooms = 8

);

### Query 2:

#### Description of logic:

Find the average cleaning review score for listings with TV.

We look at the property amenities which contains a TV and we make a join between Scores and Property. Then we select the average of cleaning review scores

#### SQL statement

SELECT AVG(scores.review\_scores\_cleanliness)

FROM Scores s, Property p

WHERE (p.amenities CONTAINS ‘TV’) AND (p.listing\_id = s.listing\_id));

### Query 3:

#### Description of logic:

Print all the hosts who have an available property between date 03.2019 and 09.2019.

We find the offers(listing\_id) whose date is between ’01-03-2019’ and ’01-09-2019’ by using a IN statement and then select the host\_id and host\_name to print the host.

#### SQL statement

SELECT DISTINCT h.host\_id, h.host\_name

FROM Host h

WHERE h.host\_id IN (

SELECT o.host\_id

FROM Offer o

WHERE o.listing\_id IN (

SELECT c.listing\_id

FROM Calendar c

WHERE c.date BETWEEN ’01-03-2019’ AND ’01-09-2019’

)

);

### Query 4:

#### Description of logic:

Print how many listing items exist that are posted by two different hosts but the hosts have the same name.

We find all the host\_id from Offer(using IN stement) by creating two tables of hosts and then keeping rows where the id’s are different but the names are the same.

#### SQL statement

SELECT COUNT(DISTINCT o.listing\_id)

FROM Offer o

WHERE o.host\_id IN (

SELECT h1.host\_id

FROM Host h1, Host h2

WHERE (h1.host\_id ! h2.host\_id) AND ( h1.host\_name = h2.host\_name)

);

### Query 5:

#### Description of logic:

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

#### SQL statement

<The SQL statement>

### Query 6:

#### Description of logic:

Find all the hosts (host\_ids, host\_names) that have only one listing.

Used the COUNT statement to count the number of listing owned by a host and this number should be equal to 1 to be part of the selected data(id’s and names).

#### SQL statement

SELECT h.host\_id, h.host\_name

FROM Host h

WHERE 1 = (

SELECT COUNT(o.listing\_id)

FROM Offer o

WHERE o.host\_id = h.host\_id

);

### Query 7:

#### Description of logic:

Find the difference in the average price of listings with and without WIFI.

First, we find the average price of listings with WIFI(using CONTAINS statement).

Then, we find find the average price of listings without WIFI(using NOT CONTAINS statement).

Finally, we do the difference between these two values.

#### SQL statement

SELECT AVG(p.price)

FROM Prices p

WHERE p.listing\_id IN (

SELECT property.listing\_id

FROM Property property

WHERE property.accomodates CONTAINS WIFI

)

-

SELECT AVG(p.price)

FROM Prices p

WHERE p.listing\_id IN (

SELECT p.listing\_id

FROM Property p

WHERE p.accomodates NOT CONTAINS WIFI

);

### Query 8:

#### Description of logic:

How much more (or less) costly to rent a room with 8 beds in Berlin compared to Madrid on average?

First, we find the average cost to rent a property with 8 bedrooms in Berlin.

We do this by finding the properties in Berlin and so having the property\_id. Then we can find the prices for these properties and so do the average.

Then we do the same for a property this time in Madrid.

Finally, we do the difference between these two results.

#### SQL statement

SELECT AVG(p.price)

FROM Prices p

WHERE p.listing\_id IN (

SELECT property.listing\_id

FROM Property property

WHERE (property.bedrooms = 8) AND property\_id IN (

SELECT a.property\_id

FROM Address a, Country c

WHERE (c.country = ‘Berlin’) AND (c.country\_id = a.country\_id)

)

)

-

SELECT AVG(p.price)

FROM Prices p

WHERE p.listing\_id IN (

SELECT property.listing\_id

FROM Property property

WHERE (property.bedrooms = 8) AND property\_id IN (

SELECT a.property\_id

FROM Address a, City c

WHERE (c.city = ‘Madrid) AND (c.city\_id = a.city\_id)

)

);

### Query 9:

#### Description of logic:

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

#### SQL statement

### Query 10:

#### Description of logic:

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

#### SQL statement

SELECT o.listing\_id, o.name

FROM Offer o

WHERE o.listing\_id IN (

SELECT p.listing\_id

FROM Property p, Property\_type pt

WHERE (p.property\_type\_id = pt.property\_type\_id) AND (pt.property\_type = ‘Apartment’) AND p.property\_id IN (

SELECT a.property\_id

FROM Address a, City c

WHERE (a.city\_id = c.city\_id) AND c.city = ‘Barcelona’

)

);

9/

SELECT h.host\_id, h.host\_name

FROM Host h, Offer o

WHERE h.host\_id = o.host\_id

GROUP BY o.host\_id

ORDER BY o.listing\_id DESC

LIMITS 10;

SELECT column1, column2, hit\_pages,...

FROM YourTable

ORDER BY hit\_pages DESC

LIMIT 5

SELECT \* FROM

(SELECT \* FROM FooTable ORDER BY X DESC LIMIT 10) as myAlias

ORDER BY X ASC

## 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>

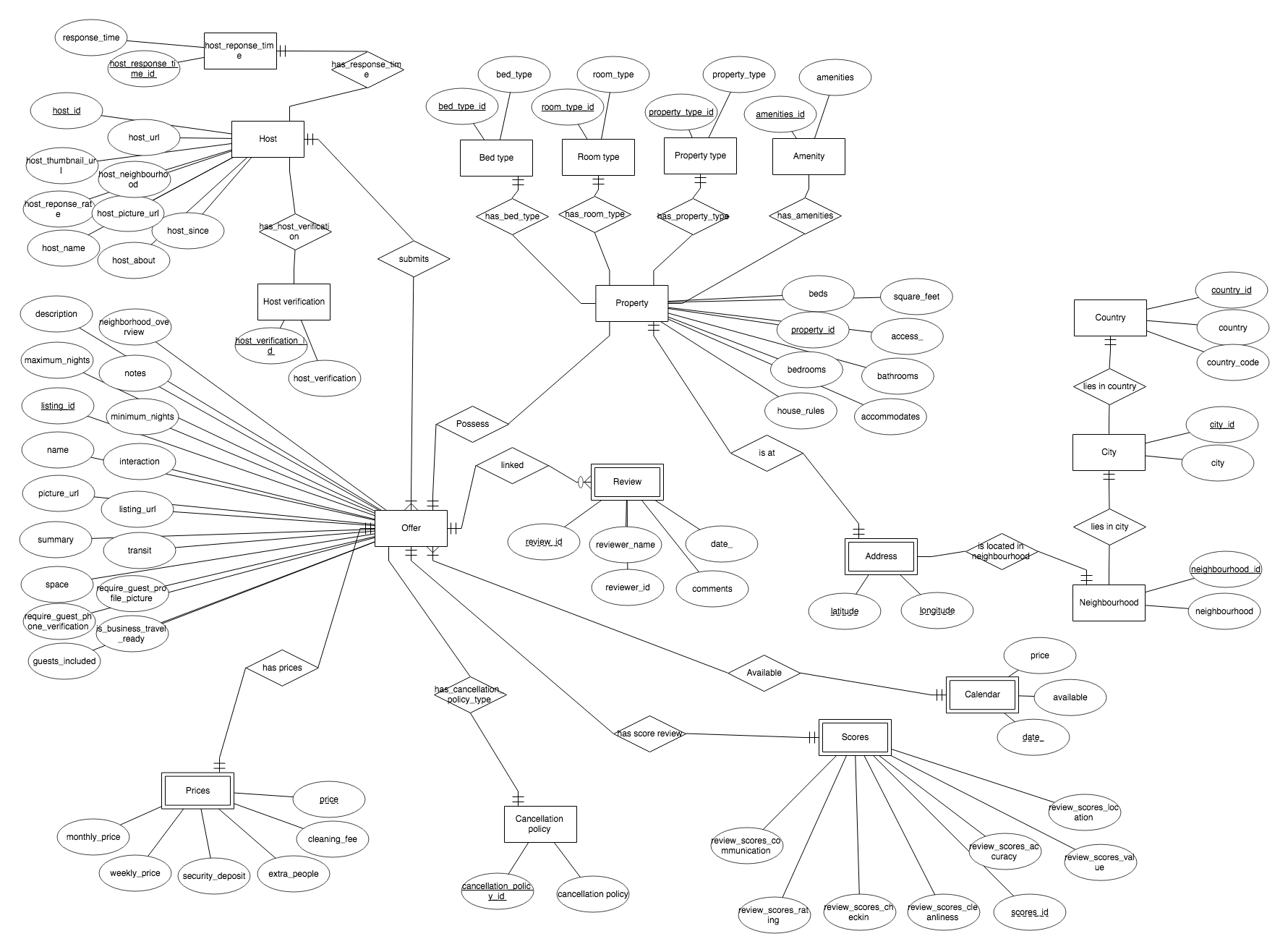
# Deliverable 3

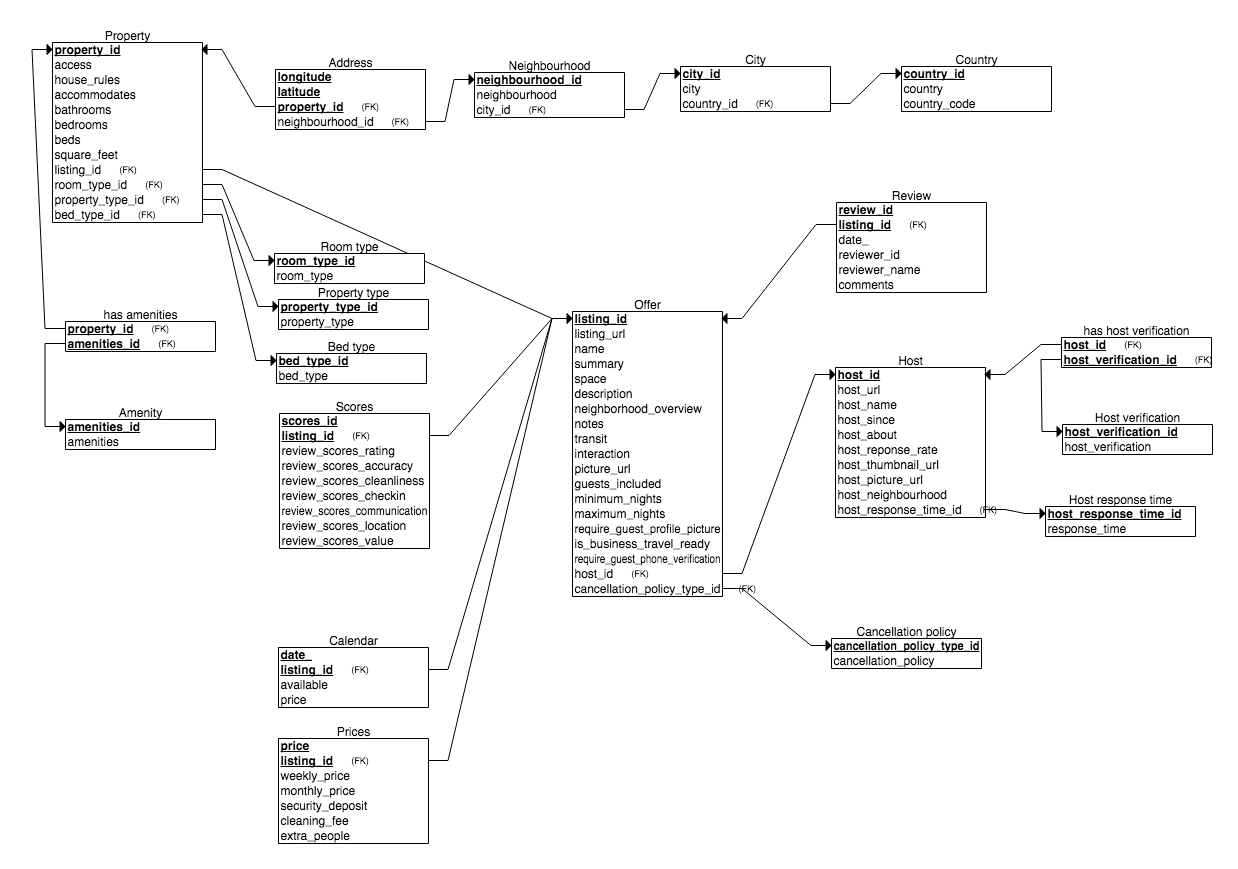
# Assumptions

First, we forgot to say that we assumed the id of a listing from the listings files was the same than the listing\_id from the reviews files. Consequently, we deleted the attribute listing\_id in the review.

We also assumed that all the ids are uniquely defined.

Then, we have assumed that the reviewer\_id is different from the host\_id even if it’s the same user.





We have translated the different relations as usual. We have grouped:

* the “Host” entity with the “has\_reponse\_time” and “neighbourhood” relationships set,
* the “Property” entity with the “has\_bed\_type”, “has\_room\_type”, “has\_property\_type” relationship set,
* the “City” entity with the “lies in country” relationship,
* “Cancellation policy“ with “has\_cancellation\_policy\_type”
* the “Neighbourhood” entity with the “lies in city” relationship.
* The “Address” with “is located in neighbourhood” relationship set.
* The “Property” entity with the “possess” relationship (because a property can have more than listing and a listing has only one property)

By doing this way and adding FOREIGN KEY from the grouped table to the other entity linked, every “one to many” relationship with optional participation are satisfied which is the case above.

We also grouped the “Offer” entity with the “submits” relationship and the “Review” entity with the “linked” relationship set. Consequently, we were not able to capture the fact that the host has mandatory participation in “submits” relationship set or that the “Offer” has mandatory participation in “linked” relationship set.

Furthermore, we had one to one relationships and so we grouped:

* “Prices” with “has prices”
* “Scores” with “has\_score review”
* “Calendar” with “available”
* “Address” with “is at”

In all these cases we were able to capture one of the two “exactly one” relationship. For example between “Offer” and “Requirements” entities, by grouping “Review” with the “has requirements” relationship set we are not able to capture the fact that an offer has mandatory participation in “has requirements” relationship set.

Finally, “Prices”, “Scores”, “Calendar”, “Address”, “Review” are all Weak Entities, we therefore added the keyword ON DELETE CASCADE after the foreign to the parent in order to make sure that the weak entity is deleted when the parent entity is deleted.

For a more general explanation of the schema’s structure:

We designed Neighbourhood, City and Country in a hierarchical way because it is more intuitive. In fact, a Neighbourood is in a City which himself is in a Country.

We normalized all predefined arguments so that queries are simpler to implement.

In fact, we normalized the bed\_type, the room\_type, the property\_type, the cancellation\_policy and their relations are one to manyvbecause for example: “property\_type” 🡪 a property have only one property type but a given property\_type can appears in multiple property.

We also normalized host\_verification and amenities and these are many to many relations.

For example: “amenities” 🡪 a property can have multiple amenities (WIFI, TV…) and a amenities(WIFI) can appear in multiple property.

**security\_deposit INT**,  
 **cleaning\_fee INT**,  
 **extra\_people INT**,  
 **listing\_id INT NOT NULL**,  
 **PRIMARY KEY** (**price**, **listing\_id**),  
 **FOREIGN KEY** (**listing\_id**) **REFERENCES** Offer(**listing\_id**) **ON DELETE CASCADE**);  
  
**CREATE TABLE** Review  
(  
 **review\_id INT NOT NULL**,  
 **date\_ DATE**,  
 **reviewer\_id INT NOT NULL**,  
 **reviewer\_name VARCHAR2**(40),  
 **comments CLOB**,  
 **listing\_id INT NOT NULL**,  
 **PRIMARY KEY** (**review\_id CREATE TABLE** Bed\_type  
(  
 **bed\_type\_id INT NOT NULL**,  
 **bed\_type VARCHAR2**(15) **NOT NULL**,  
 **PRIMARY KEY** (**bed\_type\_id**)  
);  
  
**CREATE TABLE** Room\_type  
(  
 **room\_type\_id INT NOT NULL**,  
 **room\_type VARCHAR2**(15) **NOT NULL**,  
 **PRIMARY KEY** (**room\_type\_id**)  
);  
  
**CREATE TABLE** Property\_type  
(  
 **property\_type\_id INT NOT NULL**,  
 **property\_type VARCHAR2**(22) **NOT NULL**,  
 **PRIMARY KEY** (**property\_type\_id**)  
);  
  
**CREATE TABLE** Cancellation\_policy  
(  
 **cancellation\_policy\_id INT NOT NULL**,  
 **cancellation\_policy VARCHAR2**(30),  
 **PRIMARY KEY** (**cancellation\_policy\_id**)  
);  
  
**CREATE TABLE** Neighbourhood  
(  
 **neighbourhood\_id INT NOT NULL**,  
 **neighbourhood VARCHAR2**(40) **NOT NULL**,  
 **PRIMARY KEY** (**neighbourhood\_id**)  
);  
  
**CREATE TABLE** Host\_response\_time  
(  
 **host\_response\_time\_id INT NOT NULL**,  
 **response\_time VARCHAR2**(20) **NOT NULL**,  
 **PRIMARY KEY** (**host\_response\_time\_id**)  
);  
  
**CREATE TABLE** Host  
(  
 **host\_id INT NOT NULL**,  
 **host\_url VARCHAR2**(45),  
 **host\_name VARCHAR2**(40),  
 **host\_since DATE**,  
 **host\_about CLOB**,  
 **host\_reponse\_rate INT**,  
 **host\_thumbnail\_url VARCHAR2**(110),  
 **host\_picture\_url VARCHAR2**(110),  
 **host\_neighbourhood VARCHAR2**(165),  
 **neighbourhood\_id INT NOT NULL**,  
 **host\_response\_time\_id INT NOT NULL**,  
 **PRIMARY KEY** (**host\_id**),  
 **FOREIGN KEY** (**host\_response\_time\_id**) **REFERENCES** Host\_response\_time(**host\_response\_time\_id**)  
);

**CREATE TABLE** has\_host\_verification  
(**PRIMARY KEY** (**host\_id**, **host\_verification\_id**),  
 **FOREIGN KEY** (**host\_id**) **REFERENCES** Host(**host\_id**),  
 **FOREIGN KEY** (**host\_verification\_id**) **REFERENCES** Host\_verification(**host\_verification\_id**);

**CREATE TABLE** Host\_verification  
(  
 **host\_verification\_id INT NOT NULL**,  
 **host\_verification VARCHAR2**(45),  
 **PRIMARY KEY** (**host\_verification\_id**)   
);

**CREATE TABLE** Offer  
(  
 **listing\_url VARCHAR2**(40),  
 **name VARCHAR2**(140),  
 **summary VARCHAR2**(1000),  
 **space VARCHAR2**(1000),  
 **listing\_id INT NOT NULL**,  
 **cancellation\_policy\_id INT NOT NULL**,  
 **description VARCHAR2**(1000),  
 **neighborhood\_overview VARCHAR2**(1000),  
 **notes VARCHAR2**(1000),  
 **transit VARCHAR2**(1000),  
 **interaction VARCHAR2**(1000),  
 **picture\_url VARCHAR2**(120),  
 **guests\_included INT**,  
 **minimum\_nights INT**,  
 **maximum\_nights INT**,  
 **host\_id INT NOT NULL**,  
 **is\_business\_travel\_ready CHAR**(1),  
 **require\_guest\_profile\_picture CHAR**(1),  
 **require\_guest\_phone\_verification CHAR**(1),  
 **PRIMARY KEY** (**listing\_id**),  
 **FOREIGN KEY** (**host\_id**) **REFERENCES** Host(**host\_id**),  
 **FOREIGN KEY** (**cancellation\_policy\_id**) **REFERENCES** Cancellation\_policy(**cancellation\_policy\_id**)  
);  
  
**CREATE TABLE** Score  
(  
 **review\_scores\_rating INT**,  
 **review\_scores\_accuracy INT**,  
 **review\_scores\_cleanliness INT**,  
 **review\_scores\_checkin INT**,  
 **review\_scores\_communication INT**,  
 **review\_scores\_location INT**,  
 **review\_scores\_value INT**,  
 **scores\_id INT NOT NULL**,  
 **listing\_id INT NOT NULL**,  
 **PRIMARY KEY** (**scores\_id**, **listing\_id**),  
 **FOREIGN KEY** (**listing\_id**) **REFERENCES** Offer(**listing\_id**) **ON DELETE CASCADE**);  
  
  
**CREATE TABLE** Property  
(  
 **access\_ VARCHAR2**(1001),  
 **house\_rules VARCHAR2**(1001),  
 **accommodates INT**,  
 **bathrooms FLOAT**,  
 **bedrooms INT**,  
 **beds INT**,  
 **square\_feet INT**,  
 **property\_id INT NOT NULL**,  
 **listing\_id INT NOT NULL**,  
 **room\_type\_id INT NOT NULL**,  
 **property\_type\_id INT NOT NULL**,  
 **bed\_type\_id INT NOT NULL**,  
 **PRIMARY KEY** (**property\_id**),  
 **FOREIGN KEY** (**listing\_id**) **REFERENCES** Offer(**listing\_id**),  
 **FOREIGN KEY** (**room\_type\_id**) **REFERENCES** Room\_type(**room\_type\_id**),  
 **FOREIGN KEY** (**property\_type\_id**) **REFERENCES** Property\_type(**property\_type\_id**),  
 **FOREIGN KEY** (**bed\_type\_id**) **REFERENCES** Bed\_type(**bed\_type\_id**)  
);

**CREATE TABLE** has\_amenities  
(**PRIMARY KEY** (**property\_id**, **amenities\_id**),  
 **FOREIGN KEY** (**property\_id**) **REFERENCES** Property(**property\_id**),  
 **FOREIGN KEY** (**amenities \_id**) **REFERENCES** Amenities(**amenities\_id**);

**CREATE TABLE** Amenity  
(  
 **amenities\_id INT NOT NULL**,  
 **amenities VARCHAR2**(1389),   
 **PRIMARY KEY** (**amenities\_id**)  
);  
  
  
**CREATE TABLE** Country  
(  
 **country\_id INT NOT NULL**,  
 **country VARCHAR2**(20) **NOT NULL**,  
 **country\_code CHAR**(2) **NOT NULL**,  
 **PRIMARY KEY** (**country\_id**)  
);  
  
**CREATE TABLE** City  
(  
 **city\_id INT NOT NULL**,  
 **city VARCHAR2**(30) **NOT NULL**,  
 **PRIMARY KEY** (**city\_id**)  
);  
  
**CREATE TABLE** Address  
(  
 **longitude FLOAT NOT NULL**,  
 **latitude FLOAT NOT NULL**,  
 **property\_id INT NOT NULL**,  
 **country\_id INT NOT NULL**,  
 **city\_id INT NOT NULL**,  
 **neighbourhood\_id INT NOT NULL**,  
 **PRIMARY KEY** (**longitude**, **latitude**, **property\_id**),  
 **FOREIGN KEY** (**country\_id**) **REFERENCES** Country(**country\_id**),  
 **FOREIGN KEY** (**city\_id**) **REFERENCES** City(**city\_id**),  
 **FOREIGN KEY** (**neighbourhood\_id**) **REFERENCES** Neighbourhood(**neighbourhood\_id**),  
 **FOREIGN KEY** (**property\_id**) **REFERENCES** Property(**property\_id**) **ON DELETE CASCADE**);

**CREATE TABLE** Prices  
(  
 **price INT NOT NULL**,  
 **weekly\_price INT**,  
 **monthly\_price INT**,  
 **listing\_id NOT NULL**,  
  
 **security\_deposit INT**,  
 **cleaning\_fee INT**,  
 **extra\_people INT**,  
 **PRIMARY KEY** (**price, listing\_id**),  
 **FOREIGN KEY** (**listing\_id**) **REFERENCES** Offer(**listing\_id**)  
 **ON DELETE CASCADE**  
);

**CREATE TABLE** Calendar  
(  
 **date\_ DATE NOT NULL**,  
 **available CHAR**(1),  
 **price INT**,  
 **listing\_id INT NOT NULL**,  
 **PRIMARY KEY** (**date\_**, **listing\_id**),  
 **FOREIGN KEY** (**listing\_id**) **REFERENCES** Offer(**listing\_id**) **ON DELETE CASCADE**);

## Query Implementation

<For each query>

### Query 1:

#### Description of logic:

Print how many hosts in each city have declared the area of their property in square meters. Sort the output based on the city name in ascending order.

Join all the table with the conditions that fulfill the area is in square meters and so we look for NON NULL values.

To count for each city we used a GROUP BY statement.

#### SQL statement

<The SQL statement>

### Query 2:

#### Description of logic:

The quality of a neighborhood is defined based on the number of listings and the review score of these listings, one way for computing that is using the median of the review scores, as medians are more robust to outliers. Find the top-5 neighborhoods using median review scores of listings in Madrid. Note: Implement the median operator on your own, and do not use the available built-in operator.

#### SQL statement

### Query 3:

#### Description of logic:

Find all the hosts (host\_ids, host\_names) with the highest number of listings.

Since there can be multiple hosts hat have the maximum number of listings, we first find the number of listing for each host, then we want this number to be equal to the maximum number of listing found with a HAVING statement.

#### SQL statement

### Query 4:

#### Description of logic:

Find the 5 most cheapest Apartments (based on average price within the available dates) in Berlin available between 01-03-2019 and 30-04-2019 having at least 2 beds, a location review score of at least 8, flexible cancellation, and listed by a host with a verifiable government id.

We have to join all the concerned table, write all selection search and do IN statement to make the querry more efficient, then we GROUP BY listing\_id, order by avg\_price and then finally fetch the 5 first rows.

#### SQL statement

### Query 5:

#### Description of logic:

Each property can accommodate different number of people (1 to 16). Find the top-5 rated (review\_score\_rating) listings for each distinct category based on number of accommodated guests with at least two of these facilities: Wifi, Internet, TV, and Free street parking.

Normalizing the amenities make this querry easier.

The IN statement is like the OR of all the list of arguments.

We GROUP BY and then filter the listings where at least two of these conditions are fulfilled. From this we do the RANK OVER PARTITION by (ACCOMODATES) statement and take at each time the five first rank, that give the answer to the querry.

#### SQL statement

### Query 6:

#### Description of logic:

What are top three busiest listings per host? The more reviews a listing has, the busier the listing is.

The last inner query COUNT the number of reviews for each host(by doing a GROUP BY).

Then we do a RANK OVER PARTITION BY host\_id and order the count found in the inner query.

Finally, we take for each partition the first 3 rows.

#### SQL statement

### Query 7:

#### Description of logic:

What are the three most frequently used amenities at each neighborhood in Berlin for the listings with “Private Room” room type?

In the inner querry we GROUP BY neighbourhood and amenities to COUNT the number of time a amenity appears. Then we do a RANK OVER PARTITION BY neighbourhood and we order by the count found in the inner querry.

Finally, in the outer querry we take for each partition the 5 first element.

#### SQL statement

### Query 1:

#### Description of logic:

#### SQL statement

### Query 9:

#### Description of logic:

What is the city who has the highest number of reviews for the room types whose average number of accommodates are greater than 3.

In the last inner query we count the number of distinct reviews where the average of ACCOMODATES for that room type is greater than 3.

Then in the middle inner query we do a RANK OVER PARTITION BY room\_type\_id and order it by the count found in the inner query.

Finally, in the outer query we take each time the first row to answer to the query.

#### SQL statement

### Query 10:

#### Description of logic:

Print all the neighborhouds in Madrid which had at least 50 percent of their listings occupied in 2019 and their host has joined airbnb no later than 01.06.2017.

We first do a WITH AS statement that enables to use these information more than once.

In the inner query we first select the neighbourhoods and COUNT the DISTINCT listing\_id occupied in 2019 and have conditions base on the WITH AS statement. Then in another inner query we select the neighbourhoods and COUNT the DISTINCT listing\_id base on the WITH AS statement.

Finally, we do a join between these two queries and check the “50 percent” condition.

#### SQL statement

### Query 1:

#### Description of logic:

#### SQL statement

### Query 1:

#### Description of logic:

#### 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>

# 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>