École polytechnique fédérale de Lausanne

Introduction to Database Systems
CS-322

Database Project

Fall 2019 — Airbnb

Team 11
Jollès Eric, Mamie Robin, Montial Charline

29th April 2019

Contents

1 ER model and DDL 2

1.1 Assumptions . . . 2

1.2 Entity Relationship Schema . . . 2

1.3 Relational Schema . . . 3

1.3.1 SQL code . . . 4

1.4 General comments . . . 8

1.4.1 Work allocation between team members . . . 8

2 First SQL Requests and Interface 9

2.1 Assumptions and Data Loading . . . 9

2.2 Query Implementation . . . 9

2.2.1 Query 1 . . . 10

2.2.2 Query 2 . . . 10

2.2.3 Query 3 . . . 11

2.2.4 Query 4 . . . 11

2.2.5 Query 5 . . . 12

2.2.6 Query 6 . . . 12

2.2.7 Query 7 . . . 13

2.2.8 Query 8 . . . 14

2.2.9 Query 9 . . . 14

2.2.10 Query 10 . . . 15

2.3 Interface . . . 15

2.3.1 Search . . . 16

2.3.2 Queries . . . 17

2.3.3 Insertion and Deletion . . . 17

2.4 General Comments . . . 18

2.4.1 Work allocation between team members . . . 18

2.4.2 Issues . . . 18

A Attributes 20

A.1 Listings . . . 20

A.2 Reviews . . . 22

A.3 Calendar . . . 23

Chapter 1

ER model and DDL

1.1 Assumptions

The majority of assumptions we made can be seen in our ER model. We made sensible assumptions about listings which will be discussed in the next section. Also, these assumptions have been made after having studied the data.

1.2 Entity Relationship Schema

In the ER-model (figure 1.1), the attribute numbers are given in appendix A. All attribute numbers not given in the ER-model belong to Listing.

First of all, it was pretty clear that the central entity would be the `Listing`. The `Listing` must be owned by exactly one `Host` entity and must be situated in precisely one `Neighbourhood`, which is also an entity.

Since it was not concretely helping nor useful to split the large amount of attributes for the entity `Listing`, they are regrouped.

Moreover, while implementing the required queries, we decided to have the amenities in a separate list. Thus we now have an `Amenity` table. Moreover, to establish the link between a particular listing and its amenities, a table `has_amenity` has been created.

Based on the same model, the entity `Host` is also linked to a new table `Host_verifications` through the table `has_host_verifications`.

The entities `City`, `Cancellation_policy`, `Room_type`, `Property_type`, `Country` and `Bed_type` have been added to avoid repetitions through the listings which may refer to the same policy, for example.

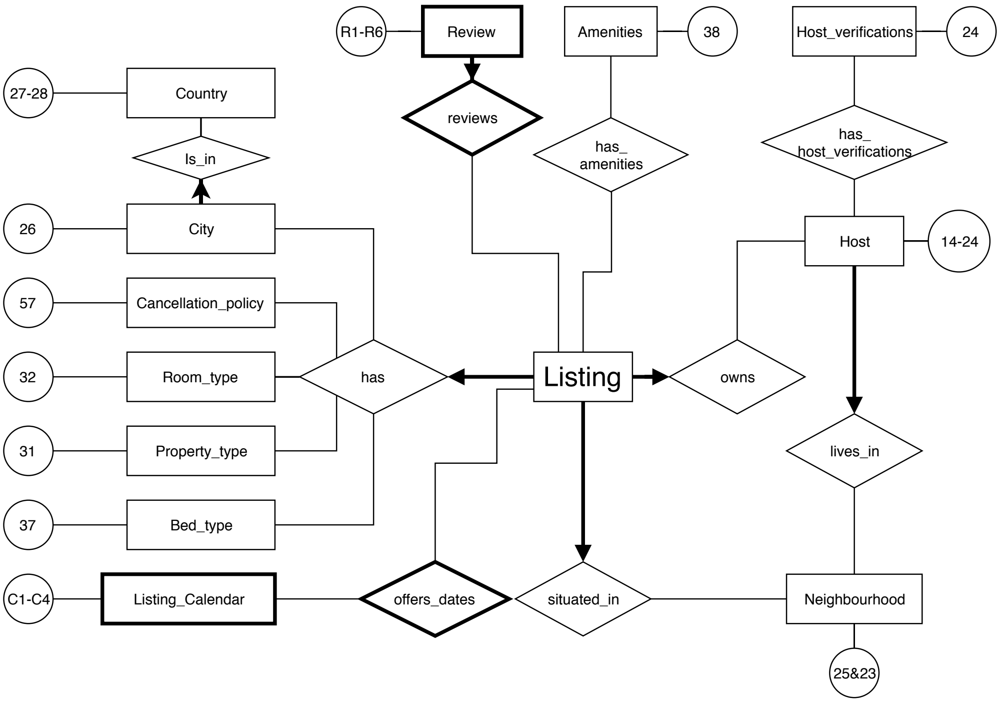


Figure 1.1: The ER model of our project.

A Host may not have any listing and can still remain in the database system but must live in exactly one Neighbourhood.

A Review must refer to exactly one Listing, otherwise it has no purpose.

Finally, the entities Country and City have been linked with a relationship is_in. The constraint is that a City must be in exactly one Country, which is indeed sensible.

1.3 Relational Schema

First, a script in Python has been written – length_finder.ipynb, which computes the maximum length of each field. We did not want to reserve a space of 1000 characters if the maximum length of this field was only of, for example, 300 characters.

About the code to create SQL tables, we naturally did our best to enforce all conditions of our ER model and use sensible types. For instance, the DATE type has been used for dates, INTEGER for ids and FLOAT for prices. Also, instead of CHAR fields, we put VARCHAR2 ones for better performance.

As previously stated, we observed that in some fields, some values were repeated in different rows. We thus decided to normalise by creating some linked entities in the database. The fields chosen are:

- host_response_time
- room_type
- property_type
- bed_type
- cancellation_policy
- Neighbourhood
- City
- Country (which contains the country and the country code)
- Amenities
- Host_Verifications

The last two fields, since they are lists, also have an additional table which corresponds to a relationship respectively between Listing and Amenities and between Host and Host_Verifications.

Concretely, the relation ship is_in represented in the ER model linking City and Country is only a foreign key – country_id in the table City.

Moreover, for all weak entities which compose the listing as stated in the ER model figure, we added the condition ON DELETE CASCADE. Indeed, all these weak entities must be deleted in case the listing they compose is deleted. These entities are Review and Listing_calendar.

1.3.1 SQL code

```
CREATE TABLE neighbourhood (
    nid              INTEGER,
    neighbourhood    VARCHAR2(40),
    PRIMARY KEY ( nid )
);

CREATE TABLE country (
    country_id      INTEGER,
    country         VARCHAR2(7),
    country_code    CHAR(2),
    PRIMARY KEY ( country_id )
);

CREATE TABLE city (
    city_id       INTEGER,
    city          VARCHAR2(40),
    country_id    INTEGER,
    PRIMARY KEY ( city_id ),
    FOREIGN KEY ( country_id )
        REFERENCES country ( country_id )
);

CREATE TABLE bed_type(
    btid         INTEGER,
    bed_type     VARCHAR2(13),
    PRIMARY KEY ( btid )
);
```

```
CREATE TABLE cancellation_policy (
    cpid INTEGER,
    cancellation_policy   VARCHAR2(27),
   PRIMARY KEY ( cpid )
);

CREATE TABLE host_response_time (
    hrtid                 INTEGER,
    host_response_time    VARCHAR2(18),
   PRIMARY KEY ( hrtid )
);

CREATE TABLE property_type (
    ptid             INTEGER,
    property_type    VARCHAR2(22),
   PRIMARY KEY ( ptid )
);

CREATE TABLE room_type (
    rtid         INTEGER,
    room_type    VARCHAR2(15),
   PRIMARY KEY ( rtid )
);

CREATE TABLE host (
    host_id          INTEGER,
    host_name        VARCHAR2(40),
    url              VARCHAR2(43),
    since            DATE,
    about            VARCHAR2(4000),
    response_time    INTEGER,
    response_rate    INTEGER,
    thumbnail_url    VARCHAR2(120),
    picture_url      VARCHAR2(120),
    nid              INTEGER,
    verifications    VARCHAR2(170),
   PRIMARY KEY ( host_id ),
   FOREIGN KEY ( response_time )
       REFERENCES host_response_time ( hrtid ),
   FOREIGN KEY ( nid )
       REFERENCES neighbourhood
);

CREATE TABLE listing (
    id                                 INTEGER,
    listing_url                        VARCHAR2(40),
    name                               VARCHAR2(150),
    summary                            VARCHAR2(1500),
    space                              VARCHAR2(1500),
    description                        VARCHAR2(1500),
    neighborhood_overview              VARCHAR2(1500),
    notes                              VARCHAR2(1500),
    transit                            VARCHAR2(1500),
    l_access                           VARCHAR2(1500),
    interaction                        VARCHAR2(1500),
    house_rules                        VARCHAR2(1500),
    picture_url                        VARCHAR2(120),
    host_id                             INTEGER,
    —neighbourhood_id
    nid                                 INTEGER,
    —city_id
    city_id                             INTEGER,
    latitude                            FLOAT,
    longitude                           FLOAT,
    —property_type_id
    ptid                                INTEGER,
    —room_type_id
    rtid                                INTEGER,
    accommodates                        INTEGER,
    bathrooms                           FLOAT,
    bedrooms                            INTEGER,
    beds                                INTEGER,
    —bed_type id
    btid                                INTEGER,
    square_feet                         INTEGER,
    price                               FLOAT,
    weekly_price                        FLOAT,
    monthly_price                       FLOAT,
    security_deposit                    FLOAT,
    cleaning_fee                        FLOAT,
    guests_included                     INTEGER,
    extra_people                        FLOAT,
    minimum_nights                      INTEGER,
    maximum_nights                      INTEGER,
    review_scores_rating                INTEGER,
    review_scores_accuracy              INTEGER,
    review_scores_cleanliness           INTEGER,
    review_scores_checkin               INTEGER,
    review_scores_communication         INTEGER,
    review_scores_location              INTEGER,
    review_scores_value                 INTEGER,
    is_business_travel_ready            CHAR(1),
    —cancellation_policy_id
    cpid                                INTEGER,
    require_guest_profile_picture       CHAR(1),
    require_guest_phone_verification    CHAR(1),
    PRIMARY KEY ( id ),
    FOREIGN KEY ( city_id )
        REFERENCES city ( city_id ),
    FOREIGN KEY ( host_id )
        REFERENCES host ( host_id ),
    FOREIGN KEY ( ptid )
        REFERENCES property_type ( ptid ),
    FOREIGN KEY ( rtid )
        REFERENCES room_type ( rtid ),
    FOREIGN KEY ( btid )
        REFERENCES bed_type ( btid ),
    FOREIGN KEY ( cpid )
        REFERENCES cancellation_policy ( cpid ),
    FOREIGN KEY ( nid )
        REFERENCES neighbourhood ( nid )
);

CREATE TABLE review (
    rid                 INTEGER,
    listing_id       INTEGER NOT NULL,
    reviewer_id      INTEGER,
    reviewer_name    VARCHAR2(60),
    rdate            DATE,
    comments         VARCHAR2(4000),
    PRIMARY KEY ( rid ),
    FOREIGN KEY ( listing_id )
        REFERENCES listing ( id )
            ON DELETE CASCADE
);

CREATE TABLE listing_calendar (
    listing_id    INTEGER,
    cdate         DATE,
    available     CHAR(1),
    price         FLOAT,
    FOREIGN KEY ( listing_id )
        REFERENCES listing ( id )
            ON DELETE CASCADE
);

CREATE TABLE amenity (
    aid         INTEGER,
    amenity     VARCHAR2(50),
    PRIMARY KEY ( aid )
);

CREATE TABLE host_verification (
    hvid                   INTEGER,
    host_verification      VARCHAR2(30),
    PRIMARY KEY ( hvid )
);

CREATE TABLE has_host_verification (
    listing_id    INTEGER,
    hvid          INTEGER,
    FOREIGN KEY ( listing_id )
        REFERENCES listing ( id )
            ON DELETE CASCADE,
    FOREIGN KEY ( hvid )
        REFERENCES host_verification ( hvid )
            ON DELETE CASCADE
);

CREATE TABLE has_amenity (
    listing_id    INTEGER,
    aid           INTEGER,
    FOREIGN KEY ( listing_id )
        REFERENCES listing ( id )
            ON DELETE CASCADE,
    FOREIGN KEY ( aid )
        REFERENCES amenity ( aid )
            ON DELETE CASCADE
);
```

1.4 General comments

1.4.1 Work allocation between team members

We naturally began to work on the ER model and what we did is that every team member had to present an ER model. The aim of this was to discuss differences we had and take the best out of the three versions. Then, for the tables, Eric wrote the `length_finder` script in Python. About the DDL code, the work has been split between Robin and Charline and the code has been mutually improved. Each of us have also contributed to writing this report.

Chapter 2

First SQL Requests and Interface

2.1 Assumptions and Data Loading

We also changed the type of some fields, indeed we converted the percent values and prices (in $) into `float`s. We also decided to drop all rows in calendar which had null values in prices since they are useless when we will make queries.

Furthermore, to save storage, we decided to only keep sub-strings for some fields, since they often are description, with redundant information (often contains translations) and not useful for queries. Since not many elements exceed these limits, it is not a big issue. These fields are:

- In `Listing`: `Neighbourhood_overview`
- In `Review`: `comments`

We made the assumption that we cannot always know all neighbourhoods' cities and countries (since we don't know the city and country of an `host_Neighbourhood`, thus we separated cities and Neighbourhood into two entities.

We also considered that all `Listing`s in a file belongs to the file `City`; we made this assumption since there was too many different city names (some with typos) in the same file.

2.2 Query Implementation

For all queries which implied prices, we considered the prices of all listings, available or not.

2.2.1 Query 1

Description of logic

We are looking for the average price of all listings which have 8 bedrooms. We solved this by using the key word AVG and adding the condition enforcing that the listing contains 8 bedrooms.

SQL statement

```
SELECT AVG (l.price)
FROM Listing l
WHERE l.bedrooms = 8
```

Result

The average price is:

```
313.15384615384615384615384615384615384615
```

Rounded:

```
313.15
```

2.2.2 Query 2

Description of logic

The query looks for all listings which propose a TV and computes the average cleaning review of this selection. We only looked for the keyword TV in amenities. Even though it can be stated in the small or longer description that there is a television, it is strictly needed to be specified in amenities by definition. This is why we only considered this field. This was our assumption to solve this query. Since Amenity is a list which contains all available amenities, we had to see if TV was part of the amenities of the listing and to establish the link between the listing and the amenities, Has_amenity has been used.

SQL statement

```
SELECT AVG(L.review_scores_cleanliness)
FROM Listing L,
   Has_amenity H,
   Amenity A
WHERE A.amenity   = 'TV'
AND H.aid           = A.aid
AND H.listing_id  = L.id
```

Result

The average cleaning review score is:

```
9.39829535417770711888358947182476594241
```

Rounded:

```
9.40
```

2.2.3 Query 3

Description of logic

The query selects all the names of the hosts who have at least one listing between the provided dates. To solve this query, we retrieved the date information in the calendar table and established the link to the host table through the listing one. The dates had to be formatted correctly to be interpreted the way we wanted.

SQL statement

```
SELECT DISTINCT H.user_name
FROM Listing L, Host H
WHERE H.user_id = L.user_id
AND L.id          IN
   ( SELECT DISTINCT listing_id
    FROM Listing_calendar
    WHERE cdate >= '01.03.19'--'2019-03-01'
    AND cdate    <= '01-09-19'--'2019-09-01'
   )
```

Result

1. Antonio
2. Kristjan Y Ana
3. Mar
4. Jaume
5. Jesus

2.2.4 Query 4

Description of logic

The query counts the number of listings whose host has the same name as another host – they must be different hosts. The other host must have at least one listing. To solve this, we matched 2 pairs of `Listing` entities with `Host` entities. Once a listing with the correct condition is found, it finds the name of the host given a listing. This checks if the names of the hosts are equal even though they are not the same person.

SQL statement

```
SELECT COUNT (DISTINCT l1.id)
```

```
FROM Listing l1, Host h1, Listing l2, Host h2
WHERE l1.user_id = h1.user_id
AND   l2.user_id = h2.user_id
AND   h1.user_name = h2.user_name
AND   h1.user_id != h2.user_id
```

Result

30393 listings fulfil this condition.

2.2.5 Query 5

Description of logic

The query looks for dates of listing whose host is `Viajes Eco`. We decided to solve it by using the `Listing` to establish the link between the `Listing_calender` table and `Host` table. We then find dates of listing whose host is `Viajes Eco` – without forgetting to ensure that the listings proposed are available.

SQL statement

```
SELECT c.cdate
FROM Listing_calendar c, Listing l, Host h
WHERE c.listing_id = l.id
AND l.user_id = h.user_id
AND h.user_name = 'Viajes Eco'
AND c.available  = 't'
```

Result

1. 03.03.19
2. 02.03.19
3. 01.03.19
4. 28.02.19
5. 27.02.19

2.2.6 Query 6

Description of logic

The query finds all hosts that only have a single listing. We decided to solve it by using a nested query. We print all host ids and host names for which the number of listings is exactly equal to one.

SQL statement

```
SELECT user_id, user_name
FROM Host
```

```
WHERE user_id IN
  ( SELECT user_id FROM Listing GROUP BY user_id HAVING COUNT(*)=1
  )
```

Result

1.	431839	Xavier
2.	95585	Daniela
3.	48815	Pols
4.	509260	Dalila
5.	66419	Teresa

2.2.7 Query 7

Description of logic

It computes the subtraction between the average price of listings with Wifi minus the average price of listings without. We directly subtract the two separate queries using the amenities list as previously done for the 2nd request to solve the query.

SQL statement

```
SELECT
  (SELECT AVG(L.price)
  FROM Listing L
  WHERE L.id IN
    (SELECT H.listing_id
    FROM Has_amenity H,
      Amenity A
      WHERE A.amenity = 'Wifi'
    AND H.aid                = A.aid
    )
  ) -
  (SELECT AVG(L.price)
  FROM Listing L
  WHERE L.id NOT IN
    (SELECT H.listing_id
    FROM Has_amenity H,
      Amenity A
      WHERE A.amenity = 'Wifi'
    AND H.aid                = A.aid
    )
  ) FROM DUAL
```

Result

The difference in the average price of listings with and without Wifi is:

`3.21388138715504444404700159175862500671`

Rounded:

```
3.21
```

2.2.8 Query 8

Description of logic

It computes the difference between the average price of a listing offering 8 bedrooms in Berlin and the average price of a listing offering 8 beds in Madrid. We solved this by subtracting the two average prices, selecting all listings with 8 beds from Madrid, and then Berlin.

SQL statement

```
SELECT ABS(avg1 - avg2) FROM
   (SELECT AVG(l1.price) AS avg1
  FROM Listing l1, City c1
  WHERE l1.beds = 8
  AND l1.cid = c1.cid
  AND c1.city = 'Berlin')
, (SELECT AVG(l2.price) AS avg2
  FROM  Listing l2, City c2
  WHERE l2.beds = 8
  AND l2.cid = c2.cid
  AND c2.city = 'Madrid')
```

Result

The absolute value of the average difference is:

```
101.592615012106537530266343825665859565
```

Rounded:

```
101.59
```

2.2.9 Query 9

Description of logic

It selects the 10 hosts who have the highest number of listings in Spain. To solve this, we grouped all listings by their hosts ids. We then ordered them in a decreasing order and took the top 10. We had to perform another manipulation to not only retrieve the host ids, but also the host names.

SQL statement

```
SELECT H.user_id, H.user_name
FROM Host H
WHERE H.user_id IN (SELECT  L.user_id
FROM Listing L, City C
WHERE L.cid   = C.cid
AND C.country = 'Spain'
GROUP BY L.user_id
ORDER BY COUNT(*) DESC
FETCH FIRST 10 ROWS ONLY)
```

Result

1.	1391607	Aline
2.	28038703	Luxury Rentals Madrid
3.	32046323	Juan
4.	299462	Stay U-Nique
5.	1408525	Mad4Rent

2.2.10 Query 10

Description of logic

It selects the 10 apartments that have the best review score rating in Barcelona. We have simply selected the listings that are in Barcelona, ordered them according to their rating, and took the top 10.

SQL statement

```
SELECT  L.id, L.name
FROM Listing L, City C
WHERE L.cid = C.cid
AND C.city = 'Barcelona'
ORDER BY L.review_scores_rating DESC
FETCH FIRST 10 ROWS ONLY
```

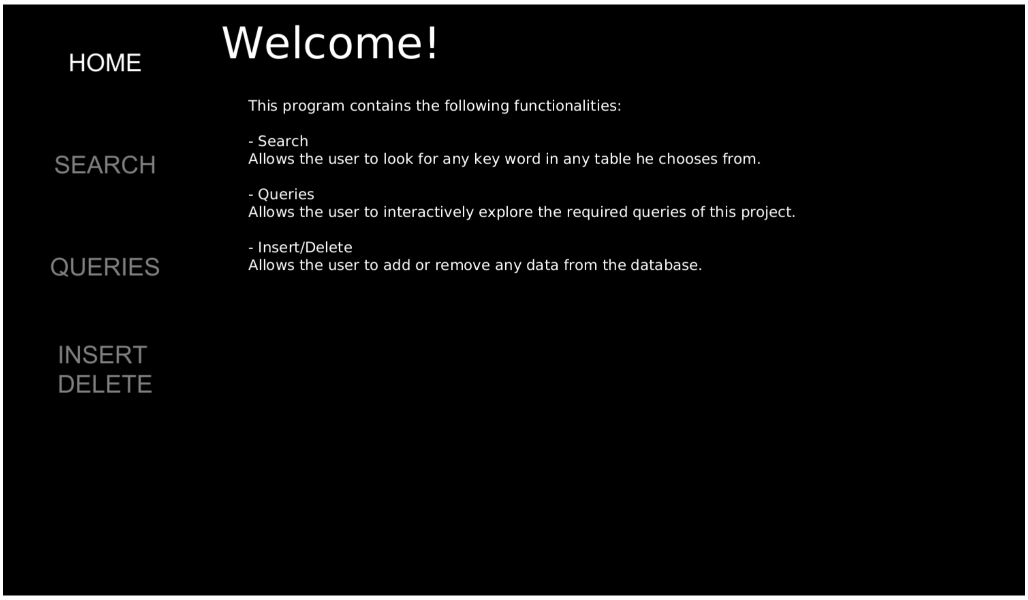
Result

1.	26033978	3 bedroom apartment in the center of barcelona}
2.	370665	vig HAPPY APARTMENT IN EL BORN
3.	6653030	Habitación mediana Poble Sec
4.	378998	PRECIOSA Y ACOGEDORA HABITACIÓN
5.	21095463	Near by parck guell

2.3 Interface

The interface was written using Scala, and mainly the ScalaFX library. The communication between the interface and the Oracle database is done using JDBC. It is composed of 4 main panels:

- **Welcome** Indicates the main functionalities of the program, purely decorative.
- **Search** Allows the user to look for any key word in any table he chooses from.
- **Queries** Allows the user to interactively explore the required queries of this project.
- **Insert/Delete** Allows the user to add or remove any data from the database.



Welcome!

This program contains the following functionalities:

- Search
Allows the user to look for any key word in any table he chooses from.

- Queries
Allows the user to interactively explore the required queries of this project.

- Insert/Delete
Allows the user to add or remove any data from the database.

Figure 2.1: The Welcome panel of the interface.

To set up the table names, the program sends the following query:

```
SELECT table_name
FROM user_tables
```

Then, to retrieve all their attributes, the program sends a query per table. Here is the request sent to retrieve the attributes of Neighbourhood.

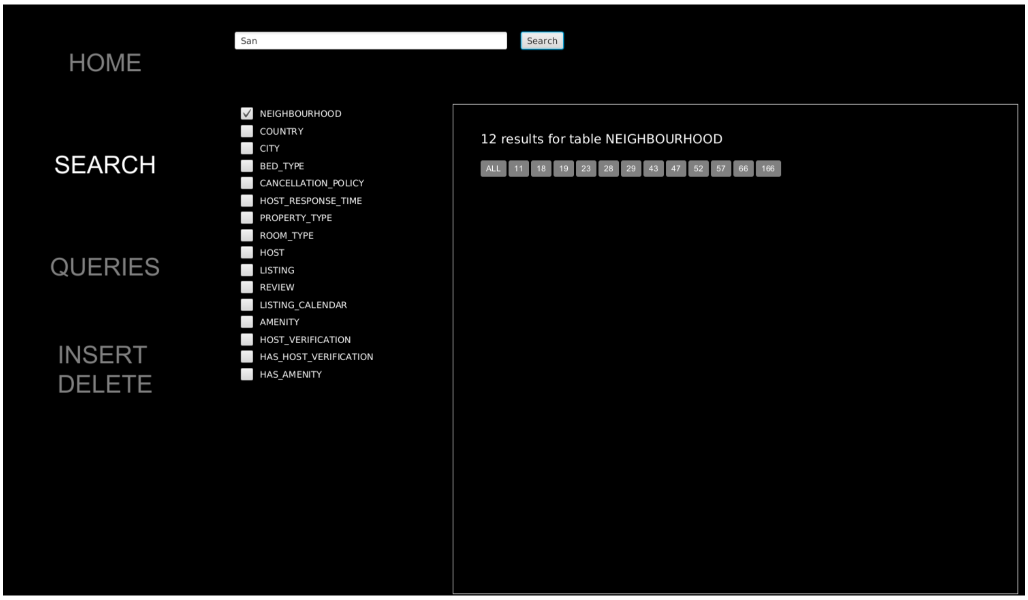
```
SELECT column_name
FROM user_tab_columns
WHERE table_name = 'NEIGHBOURHOOD'
```

2.3.1 Search

The attributes of each table are then used for the search function. Here is the query when searching for `San` in the Neighbourhood table:

```
SELECT NID
FROM NEIGHBOURHOOD
WHERE NID LIKE '%San%' OR NEIGHBOURHOOD LIKE '%San%'
```

The query selects the primary keys to then display them on the buttons, showed in figure 2.2. The keys are stored so that the buttons are functional, and an `ALL` button can display all the results at once.



☑ NEIGHBOURHOOD
☐ COUNTRY
☐ CITY
☐ BED_TYPE
☐ CANCELLATION_POLICY
☐ HOST_RESPONSE_TIME
☐ PROPERTY_TYPE
☐ ROOM_TYPE
☐ HOST
☐ LISTING
☐ REVIEW
☐ LISTING_CALENDAR
☐ AMENITY
☐ HOST_VERIFICATION
☐ HAS_HOST_VERIFICATION
☐ HAS_AMENITY

12 results for table NEIGHBOURHOOD

Figure 2.2: The Search panel of the interface.

Then, after clicking on e.g. the ALL button, the following query is sent:

```
SELECT *
FROM NEIGHBOURHOOD
WHERE   NID = '11' OR NID = '18' OR NID = '19' OR NID = '23' OR NID = '28' OR
    NID = '29' OR NID = '43' OR NID = '47' OR NID = '52' OR NID = '57' OR NID =
     '66' OR NID = '166'
```

As a side note, all these queries are built using Scala methods on `List`, assuring efficient string builds.

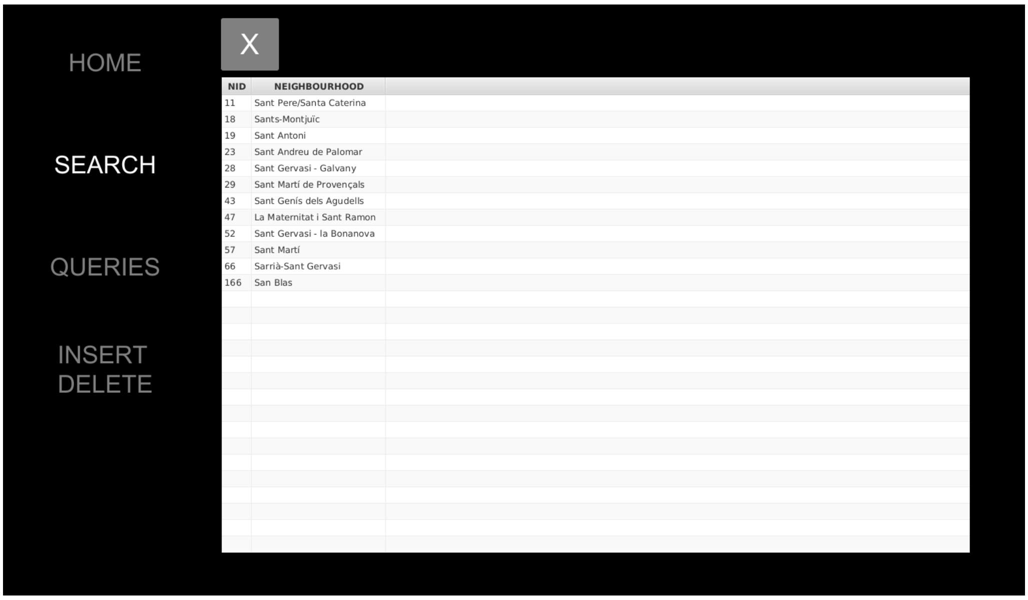
This spawns the window showed in figure 2.3.

2.3.2 Queries

Figure 2.4 shows the window allowing the user to send the predefined queries. The queries sent are the same as the ones described in section 2.2. In the future, a place to input query parameters will be designed.

2.3.3 Insertion and Deletion

Not yet implemented, but its design will be close to the other interactive windows. The right side of the window will display all the necessary fields used to create and/or delete data.



NID	NEIGHBOURHOOD
11	Sant Pere/Santa Caterina
18	Sants-Montjuïc
19	Sant Antoni
23	Sant Andreu de Palomar
28	Sant Gervasi - Galvany
29	Sant Martí de Provençals
43	Sant Genís dels Agudells
47	La Maternitat i Sant Ramon
52	Sant Gervasi - la Bonanova
57	Sant Martí
66	Sarrià-Sant Gervasi
166	San Blas

Figure 2.3: The result of the Search displayed.

2.4 General Comments

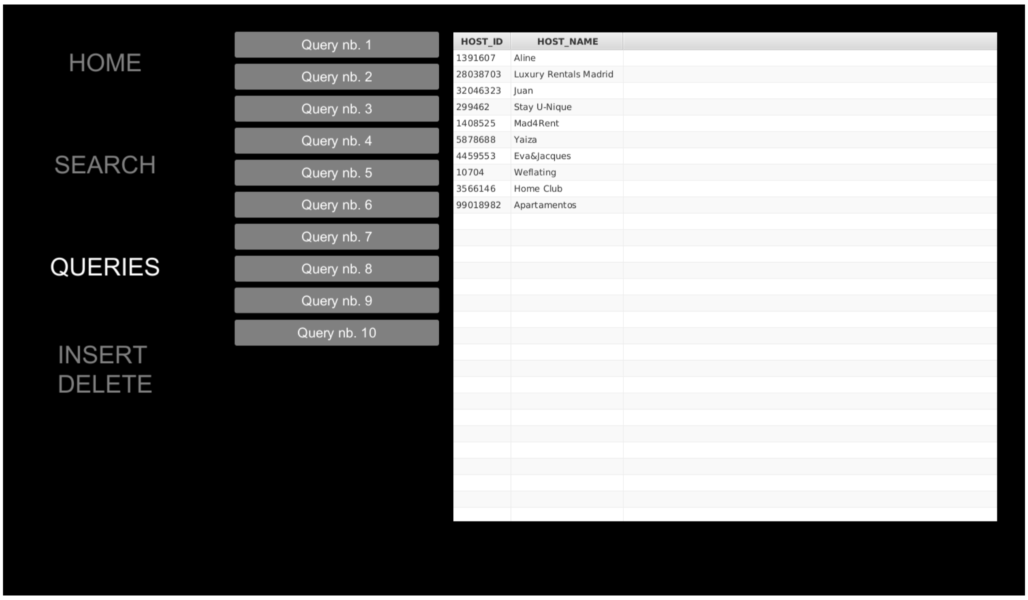
2.4.1 Work allocation between team members

Eric worked on the data parsing and its insertion into the database. Charline wrote the SQL requests, with close collaboration with all team members. Robin designed and created the graphical interface and its related SQL queries.

2.4.2 Issues

We did not manage to give the `TEXT` type to some attributes as advised in the previous feedback. SqlDevelopper returned the following error when we tried to do it:

```
SQL Error : ORA-00902: invalid datatype
```



HOST_ID	HOST_NAME
1391607	Aline
28038703	Luxury Rentals Madrid
32046323	Juan
299462	Stay U-Nique
1408525	Mad4Rent
5878688	Yaiza
4459553	Eva&Jacques
10704	Weflating
3566146	Home Club
99018982	Apartamentos

Figure 2.4: The queries panel of the interface.

Appendix A

Attributes

A.1 Listings

1. **Id** The unique listing identifier.

2. **listing_url** The URL of the listing.

3. **name** The name of the listing.

4. **summary** A small description of the listing.

5. **space** A small description of the space of the listing.

6. **description** A large description of the listing.

7. **neighborhood_overview** Description of the neighbourhood of the listing.

8. **notes** An extra note about the listing.

9. **transit** Description of the transportation to the listing.

10. **access** Specification of the accessibilities of household stuff, such as kitchen facilities.

11. **interaction** Description of whom/how to interact regarding the listing.

12. **house_rules** House rule specifications.

13. **picture_url** The URL to the picture of the listing.

14. **host_id** The unique host identifier.

15. **host_url** The URL of the host.

16. **host_name** The name of the host.

17. **host_since** The date that the host has started working with Airbnb.

18. **host_about** A small description of the host.

19. **host_response_time** The amount of time within which the host responses.

20. **host_response_rate** The rate at which the host replies the messages.

21. **host_thumbnail_url** The URL to a thumbnail profile photo of the host.

22. **host_picture_url** The URL to a profile photo of the host.

23. **host_neighbourhood** The neighbourhood the host lives in.

24. **host_verifications** The way with which the host can be verified.

25. **neighbourhood** The neighbourhood where the listing is in.

26. **city** The city where the listing is in.

27. **country_code** The code of the country where the listing is in.

28. **country** The country where the listing is in.

29. **latitude** The latitude of the listing.

30. **longitude** The longitude of the listing.

31. **property_type** The type of the property.

32. **room_type** The type of the room.

33. **accommodates** The number of people that the listing can accommodate.

34. **bathrooms** The number of bathrooms that the listing has.

35. **bedrooms** The number of bedrooms that the listing has.

36. **beds** The number of beds that the listing has.

37. **bed_type** The type of the beds.

38. **amenities** The set of amenities that listing features.

39. **square_feet** The area of the listings in square feet.

40. **price** The daily price of the listing. It is the price for the day when the data is collected. For the price for a particular date, please see the *_calendar.csv files.

41. **weekly_price** The weekly price of the listing.

42. **monthly_price** The monthly price of the listing.

43. **security_deposit** The amount of money for security deposit.

44. **cleaning_fee** The fee for cleaning.

45. **guests_included** The number of guests that the daily price covers.

46. **extra_people** The additional price to be paid for every extra guest in addition to the number of guests specified by the guests_included attribute.

47. **minimum_nights** The minimum number of nights to rent.

48. **maximum_nights** The maximum number of nights to rent.

49. **review_scores_rating** The rating score of the listing.

50. **review_scores_accuracy** The accuracy score of the listing.

51. **review_scores_cleanliness** The cleanliness score of the listing.

52. **review_scores_checkin** The checkin score of the listing (to quantify how easy the checkin is).

53. **review_scores_communication** The communication score of the host.

54. **review_scores_location** The location score of the listing.

55. **review_scores_value** The score on the value that the listing provides for the price.

56. **is_business_travel_ready** Whether the listing can be used for business travels.

57. **cancellation_policy** The cancellation policy of the listing.

58. **require_guest_profile_picture** Whether the listing requires a guest profile picture.

59. **require_guest_phone_verification** Whether the listing requires guest phone verification.

A.2 Reviews

R1. **listing_id** The identifier of the listing that is reviewed.

R2. **id** The unique review identified.

R3. **date** The date that the review has been written.

R4. **reviewer_id** The uniquer reviewer identified

R5. reviewer_name The name of the reviewer

R6. comments The review.

A.3 Calendar

C1. listing_id The identifier of the listing whose availability and price information is given.

C2. date The date on which the listing is available or not.

C3. available Whether the listing is available or not.

C4. price The price of the listing for the particular date.