Databases Project – Spring 2019

Team No: 26

Names: Camilla Giaccari, Hédi Sassi, Simon Perriard

Contents

Contents 1

Deliverable 1 4

Assumptions 4

Entity Relationship Schema 4

Schema 4

Description 4

Relational Schema 5

ER schema to Relational schema 5

DDL 5

General Comments 6

Deliverable 2 7

Assumptions 7

Data Loading 7

Query Implementation 7

Query 1: 7

Description of logic: 7

SQL statement 7

Result 7

Query 2: 8

Description of logic: 8

SQL statement 8

Result 8

Query 3: 8

Description of logic: 8

SQL statement 8

Result 9

Query 4: 9

Description of logic: 9

SQL statement 9

Result 10

Query 5: 10

Description of logic: 10

SQL statement 10

Result 10

Query 6: 11

Description of logic: 11

SQL statement 11

Result 11

Query 7: 11

Description of logic: 11

SQL statement 12

Result 12

Query 8: 12

Description of logic: 12

SQL statement 12

Result 13

Query 9: 13

Description of logic: 13

SQL statement 13

Result 14

Query 10: 14

Description of logic: 14

SQL statement 14

Result 15

Interface 15

Design logic Description 15

Screenshots 15

General Comments 16

Deliverable 3 17

Assumptions 17

Query Implementation 17

Query a: 17

Description of logic: 17

SQL statement 17

Query Analysis 17

Selected Queries (and why) 17

Query 1 17

Query 2 17

Query 3 18

Interface 18

Design logic Description 18

Screenshots 18

General Comments 18

Deliverable 1

Assumptions

We made no assumption concerning the correctness of the data, we checked every field of every CSV file. The type of each field has been checked and each line containing a wrong input (i.e. missing mandatory field, negative price,…) has been kicked out of the dataset.

We defined some mandatory fields, listed below.

Listings: listing\_id, listing\_url, listing\_name, host\_id, host\_url, host\_name

Reviews: all fields are mandatory

Calendar: all fields except price are mandatory

Entity Relationship Schema

Schema

The schema can be found here : <https://github.com/hedi-sassi/rbnb_db_project/tree/master/ER>

Description

First, we chose to separate the main listing attributes and the listing’s details. This way, when we want display a lot of listings, we only query the table with the main information (small description, name, thumbnail etc..) and if someone select the listing, we also query the tables containing the details.

We did the same for the host and host details tables.

The listing table is connected (one to one mapping) with the calendar, review scores, material description and cost details tables. This implies they are all weak entities with respect to the listing table.

We decided to create special tables to hold the amenities and the host verifications as those are list attributes. We link them to the listing using intermediate tables containing the listing id and the amenities/host verifications id.

Relational Schema

ER schema to Relational schema

The schema can be found here : <https://github.com/hedi-sassi/rbnb_db_project/tree/master/relational_model>

Weak entities are accounted for with the help of foreign keys. If the foreign key is not present, it will trigger a “Cascade” deletion policy.

DDL

The DDL can be found here: <https://github.com/hedi-sassi/rbnb_db_project/tree/master/relational_model>

General Comments

We split the work as followed:

- ER model: Camilla

- Relational Model : Simon

- Data verification (scala program on the repo) : Hédi

Deliverable 2

Assumptions

We made no assumption about the data.

The ER schema and the Relational schema have been updated following the directions given by the TAs for Milestone1.

Data Loading

We organized the data as described in the Relational schema and imported it using SQLDeveloper.

Query Implementation

Query 1:

What is the average price for a listing with 8 bedrooms?

#### Description of logic:

We take the average value of the price attribute, considering only the listings with 8 bedrooms in their material description.

#### SQL statement

select AVG(CD.PRICE)

from COSTS\_DETAILS CD

INNER JOIN MATERIAL\_DESCRIPTION M

ON CD.LISTING\_ID = M.LISTING\_ID

where M.BEDROOMS = 8;

#### Result

Only one row with value 313,153846153846153846153846153846153846

Query 2:

What is the average cleaning review score for listings with TV?

#### Description of logic:

We take the average value of the REVIEW\_SCORES\_CLEANLINESS attribute, considering only the listings with ‘TV’ in their amenities.

#### SQL statement

select AVG(RS.REVIEW\_SCORES\_CLEANLINESS)

from REVIEWS\_SCORES RS

INNER JOIN LISTING\_AMENITIES LA

ON RS.LISTING\_ID = LA.LISTING\_ID

INNER JOIN AMENITIES AM

ON AM.AMENITY\_ID = LA.AMENITY\_ID

where AM.AMENITY\_NAME = 'TV';

#### Result

Only one row with value 9,39864565813932902540497477206337965832

Query 3:

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

#### Description of logic:

We take all the informations about the hosts who have a listing with available date following (>=) 03.2019 and prior (<=) 09.2019.

#### SQL statement

select \*

from HOST H

where H.HOST\_ID IN (select L.HOST\_ID

from LISTING L, CALENDAR CA

where L.LISTING\_ID = CA.LISTING\_ID and CA.AVAILABLE = 't' and CA.CALENDAR\_DATE >= '01-MAR-19' and CA.CALENDAR\_DATE <= ’30-SEP-19' );

#### Result

(HOST\_ID; HOST\_URL; HOST\_NAME; HOST\_SINCE; HOST\_THUMBNAIL\_URL)

71615 https://www.airbnb.com/users/show/71615 Mireia And Maria 19-GEN-10 https://a0.muscache.com/im/users/71615/profile\_pic/1426612511/original.jpg?aki\_policy=profile\_small

82522 https://www.airbnb.com/users/show/82522 Meritxell 18-FEB-10 https://a0.muscache.com/im/pictures/ece65ffd-a798-4209-b1b0-a51060412b29.jpg?aki\_policy=profile\_small

108310 https://www.airbnb.com/users/show/108310 Pedro 14-APR-10 https://a0.muscache.com/im/pictures/user/7f7e9c1a-7274-4e90-a797-f079ffd9a9a3.jpg?aki\_policy=profile\_small

134698 https://www.airbnb.com/users/show/134698 Svetlana 29-MAG-10 https://a0.muscache.com/im/users/134698/profile\_pic/1334849467/original.jpg?aki\_policy=profile\_small

136853 https://www.airbnb.com/users/show/136853 Fidelio 02-GIU-10 https://a0.muscache.com/im/users/136853/profile\_pic/1312382561/original.jpg?aki\_policy=profile\_small

Query 4:

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

#### Description of logic:

We use COUNT to determine how many different listing with (IN) different host\_id having the same host\_name exist.

#### SQL statement

select COUNT(\*)

from LISTING L, HOST H

where L.HOST\_ID = H.HOST\_ID and H.HOST\_ID IN (Select H1.HOST\_ID

from HOST H1, Host H2

where H1.HOST\_NAME = H2.HOST\_NAME and H1.HOST\_ID != H2.HOST\_ID

);

#### Result

Only one row with value 30343

Query 5:

Print all the dates that 'Viajes Eco' has available accommodations for rent.

#### Description of logic:

We take the calendar\_date of all the listings with availability value ’t’ (true) and host\_name ‘Viajes Eco’.

#### SQL statement

select CA.CALENDAR\_DATE

from CALENDAR CA

INNER JOIN LISTING L

ON CA.LISTING\_ID = L.LISTING\_ID and CA.AVAILABLE = 't'

INNER JOIN HOST H

ON L.HOST\_ID = H.HOST\_ID

where H.HOST\_NAME = 'Viajes Eco';

#### Result

10-NOV-18

11-NOV-18

12-NOV-18

13-NOV-18

14-NOV-18

Query 6:

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

#### Description of logic:

We take only the ones that are in the group of hosts with only one different host\_id per listing.

#### SQL statement

select H.HOST\_ID, H.HOST\_NAME

from HOST H

where H.HOST\_ID IN (select L.HOST\_ID

from LISTING L

group by L.HOST\_ID having COUNT(\*) = 1

);

#### Result

(HOST\_ID; HOST\_NAME)

108310 Pedro

73163 Andres

158596 Ester

90417 Etain

280070 Cristina

Query 7:

What is the difference in the average price of listings with and without Wifi?

#### Description of logic:

We created a view named “wifi” (with all the listings with ‘WiFi’ in their amenities) and then used it in the SQL statement:

create view wifi as

select LA.LISTING\_ID

from AMENITIES AM, LISTING\_AMENITIES LA

where AM.AMENITY\_ID = LA.AMENITY\_ID and AM.AMENITY\_NAME = 'Wifi';

#### SQL statement

select AVG(CD1.PRICE) - AVG(CD2.PRICE)

from COSTS\_DETAILS CD1, COSTS\_DETAILS CD2

where CD1.LISTING\_ID in (select \* from wifi)

and CD2.LISTING\_ID not in (select \* from wifi);

#### Result

Only one row with value 6,66174164496683882662676775669337783597

Query 8:

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

#### Description of logic:

We take the subtraction of two average prices: the first one is from the listings with 8 beds in their material\_description and Berlin as their city; the second one is from the listings with 8 beds in their material\_description and Madrid as their city.

#### SQL statement

select AVG(CD1.PRICE) - AVG(CD2.PRICE)

from COSTS\_DETAILS CD1, COSTS\_DETAILS CD2

where CD1.LISTING\_ID IN (select MD.LISTING\_ID

from MATERIAL\_DESCRIPTION MD

where MD.BEDS = 8)

and CD1.LISTING\_ID IN (select L.LISTING\_ID

from LISTING L

where L.CITY = 'Berlin')

and CD2.LISTING\_ID not in (select MD.LISTING\_ID

from MATERIAL\_DESCRIPTION MD

where MD.BEDS = 8)

and CD2.LISTING\_ID IN (select L.LISTING\_ID

from LISTING L

where L.CITY = 'Madrid');

#### Result

Only one row with value 44,46580490444090251071110006287744746763

Query 9:

Find the top-10 (in terms of the number of listings) hosts (host\_ids, host\_names) in Spain.

#### Description of logic:

We use “order by COUNT(\*) DESC” to determine the hosts with the more listings in a descending order. We take only the listings with Spain as their country. We use “where rownum <= 10” to take only the first 10 rows of the result.

#### SQL statement

select \* from

(select H.HOST\_ID , H.HOST\_NAME

from LISTING L, HOST H

where L.COUNTRY = 'Spain' and L.HOST\_ID = H.HOST\_ID

group by L.HOST\_ID, H.HOST\_NAME, H.HOST\_ID

order by COUNT(\*) DESC)

where rownum <= 10;

#### Result

(HOST\_ID; HOST\_NAME)

4459553 Eva&Jacques

99018982 Apartamentos

32046323 Juan

28038703 Luxury Rentals Madrid

1391607 Aline

Query 10:

Find the top-10 rated apartments in Barcelona.

#### Description of logic:

As for the previous query, we use “order by RS.REVIEW\_SCORES\_RATING DESC” to determine the rating scores in a descending order. We take only the listings with Barcelona as their city and ‘Apartment’ as their type. We use “where rownum <= 10” to take only the first 10 rows of the result.

#### SQL statement

select \* from (

select L.LISTING\_ID, L.LISTING\_NAME

from LISTING L

INNER JOIN REVIEWS\_SCORES RS

ON L.LISTING\_ID = RS.LISTING\_ID and L.CITY = 'Barcelona'

INNER JOIN MATERIAL\_DESCRIPTION MD

ON MD.LISTING\_ID = L.LISTING\_ID and MD.PROPERTY\_TYPE = 'Apartment'

order by RS.REVIEW\_SCORES\_RATING DESC)

where rownum <=10;

#### Result

(LISTING\_ID; LISTING\_NAME)

475786 Room to rent in beautiful apartment

763465 Very Nice Room to rent in Raval

783032 room for rent 10 minutes from center

740113 Sunny, authentic Sant Antoni Apartment

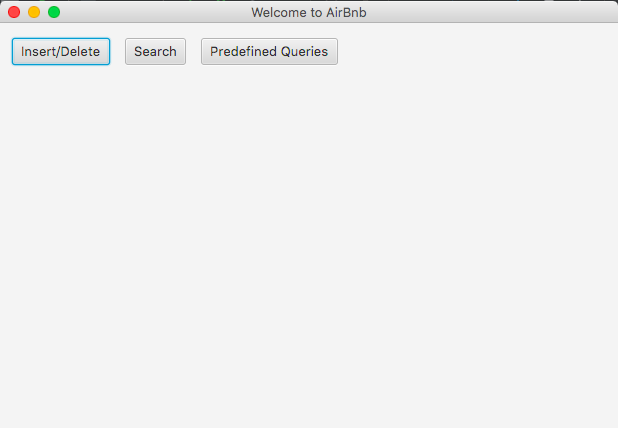
721510 32 Valencia Apartment 2 bedrooms

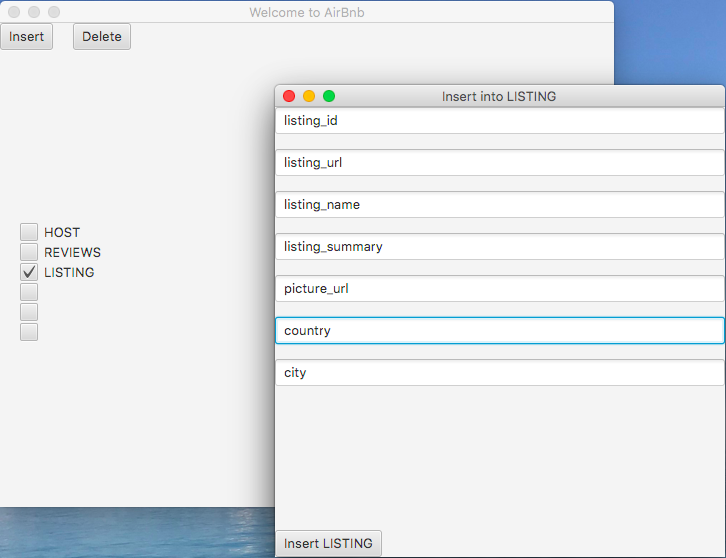
Interface

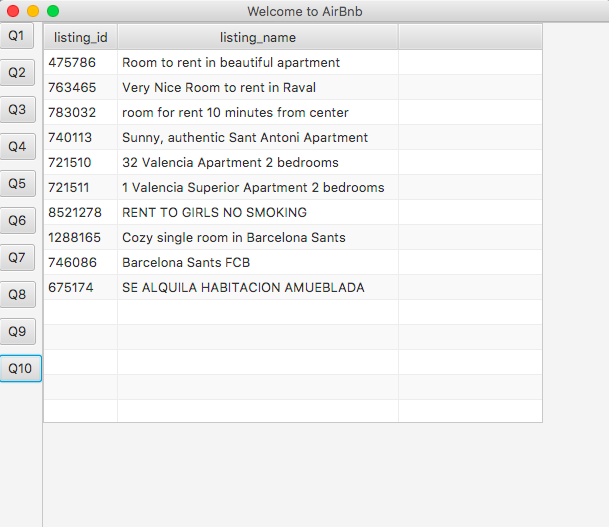
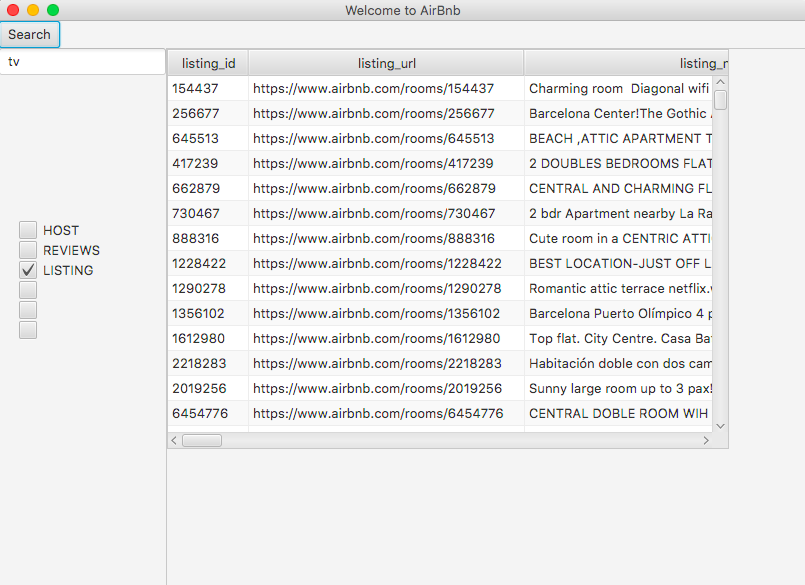
Design logic Description

We decided to use JavaFX.

The interface is simple with intuitive buttons: it’s possible to insert/delete data by modifying the attributes of the item; there is a ‘Search’ button to search for a key-word in the database; the 10 queries described above are added as ‘Predefined queries’, so that the program can give the results to the user without showing any SQL language.

Screenshots





General Comments

We split the work as followed:

- Data insertion in the DB: Simon and Hédi

- Queries writing: Simon and Hédi

* Queries test and corrections: Camilla
* User interface: Hédi
* Report: Camilla

Deliverable 3

Assumptions

No assumptions where made about the data (as in previous milestones). We wrote a parser to clean, check and regroup the data before insertion.

Query Implementation

All the queries can be found here: <https://github.com/hedi-sassi/rbnb_db_project/blob/master/DB_project.sql>

Since the queries are long, it may be better to use the ones on the git repo rather than those written below (not easily readable).

Here is our solutions for the queries of milestone 3:

Query 1:

#### Description of logic:

First we decided to find the hosts that have listing with the dimension (in square feet) not null and then we grouped those hosts by city, counted those hosts and ordered the result by city

#### SQL statement

select count(distinct(h.HOST\_ID)), l1.CITY

from HOST h, LISTING l1, MATERIAL\_DESCRIPTION md

where h.HOST\_ID = l1.HOST\_ID and l1.LISTING\_ID = md.LISTING\_ID and md.SQUARE\_FEET is not null

group by l1.CITY order by l1.CITY asc;

Result:

345 Barcelona

370 Berlin

249 Madrid

Query 2:

#### Description of logic:

First we selected all the listings in Madrid with non-null review score rating. Then, we order the listings according to their review score rating per neighborhood (using partition by) and add a column for the rownumber. We then compute the number of listings per neighborhood and we divide it by 2 so we have the median position per neighborhood.

Eventually, we filter the ordered listings to only select the ones that are in the median position per neighborhood and select the results where row number smaller than 5

#### SQL statement

#### select \* from

#### (select distinct(med\_per\_ng.NEIGHBORHOOD), REVIEW\_SCORES\_RATING from

#### (select distinct(loc.NEIGHBORHOOD), floor((count(\*) over(partition by loc.NEIGHBORHOOD)+1)/2) as median\_elem\_per\_ng

#### from REVIEWS\_SCORES rs, LISTING l, LISTING\_LOCATION loc

#### where rs.LISTING\_ID = l.LISTING\_ID and loc.LISTING\_ID = l.LISTING\_ID and rs.REVIEW\_SCORES\_RATING is not null and l.CITY = 'Madrid') med\_per\_ng,

#### (select loc.NEIGHBORHOOD, rs.REVIEW\_SCORES\_RATING, ROW\_NUMBER() over(partition by loc.NEIGHBORHOOD order by rs.REVIEW\_SCORES\_RATING desc) as rnum

#### from REVIEWS\_SCORES rs, LISTING\_LOCATION loc, LISTING l

#### where rs.LISTING\_ID = loc.LISTING\_ID and l.LISTING\_ID = rs.LISTING\_ID and l.CITY = 'Madrid' and rs.REVIEW\_SCORES\_RATING is not null) ranked\_by\_ng\_and\_rev

#### where med\_per\_ng.NEIGHBORHOOD = ranked\_by\_ng\_and\_rev.NEIGHBORHOOD and median\_elem\_per\_ng = rnum

#### order by REVIEW\_SCORES\_RATING desc)

#### where rownum <= 5

#### ;

#### Result:

#### Estrella 100

#### Tetuán 100

#### Hispanoamérica 98

#### Vallehermosa 98

#### Vicálvaro 98

Query 3:

#### Description of logic:

We counted the number of listings per host using the count function and groupy by host\_id. We then ranked thoses results and selected the lines with rank = 1 (with ties).

#### SQL statement

select h.HOST\_ID, h.HOST\_NAME

from

(select HOST\_ID, rank() over(order by nbr desc) as rnk

from

(select L.HOST\_ID, count(\*) as nbr

from LISTING l

group by l.HOST\_ID)

) ranked

,

HOST h

where h.HOST\_ID = ranked.HOST\_ID and ranked.rnk = 1;

Result:

4459553 Eva&Jacques

Query 4:

#### Description of logic:

First we filter the concerned listings, then we compute the average price using the calendar and eventually we order and take the top 5 using rownum.

#### SQL statement

#### select \* from

#### (select AVG(cal.PRICE) as average, cal.LISTING\_ID

#### from

#### (select l.LISTING\_ID from

#### LISTING l, MATERIAL\_DESCRIPTION md, REVIEWS\_SCORES rs, LISTING\_DETAILS ld

#### where l.LISTING\_ID = md.LISTING\_ID and l.LISTING\_ID = rs.LISTING\_ID and

#### l.LISTING\_ID = ld.LISTING\_ID and l.CITY = 'Berlin'

#### and md.PROPERTY\_TYPE = 'Apartment' and

#### md.BEDS >= 2 and rs.REVIEW\_SCORES\_LOCATION >= 8

#### and ld.CANCELLATION\_POLICY = 'flexible' and

#### l.HOST\_ID IN (

#### select hv.HOST\_ID

#### from HOST\_VERIFICATIONS hv, VERIFICATIONS v

#### where hv.VERIFICATION\_ID = v.VERIFICATION\_ID and v.VERIFICATION\_NAME LIKE '%government\_id%')

#### ) filtered

#### ,

#### CALENDAR cal

#### where cal.LISTING\_ID = filtered.LISTING\_ID and cal.CALENDAR\_DATE between date'2019-03-01' and date'2019-04-30' and cal.AVAILABLE = 't'

#### group by cal.LISTING\_ID order by average asc) averaged

#### where rownum <= 5;

Result:

20 1490274

21.0655738 24043706

21.2903226 1368460

22 7071541

22 6691656

Query 5:

#### Description of logic:

We filtered the amenities concerned (from the table with all the amenities for each listings) and then grouped them by listing and selected those with a counted >=2 (thus the listings will have at least 2 of the concerned amenities).

Then we just had to get their review score , partition them by the number of person they can accommodate and rank them.

Eventually we selected those with rank <=5.

#### SQL statement

#### select \* from

#### (select filtered.LISTING\_ID, md.ACCOMODATES, ROW\_NUMBER() over(partition by md.ACCOMODATES order by rs.REVIEW\_SCORES\_RATING desc) as ranked

#### from

#### (select facilities.LISTING\_ID from

#### (select la.LISTING\_ID, count(\*) as counted

#### from AMENITIES am, LISTING\_AMENITIES la

#### where la.AMENITY\_ID = am.AMENITY\_ID and

#### (am.AMENITY\_NAME = 'Wifi' or am.AMENITY\_NAME = 'Internet' or

#### am.AMENITY\_NAME = 'TV' or am.AMENITY\_NAME = 'Free street parking')

#### group by la.LISTING\_ID) facilities

#### where facilities.counted >= 2) filtered,

#### MATERIAL\_DESCRIPTION md, REVIEWS\_SCORES rs

#### where filtered.LISTING\_ID = md.LISTING\_ID and rs.LISTING\_ID = filtered.LISTING\_ID) rnk

#### where ranked <= 5

#### ;

Result:

475786 1

675175 1

675174 1

676924 1

1288165 1

539349 2

Query 6:

#### Description of logic:

First we count the number of reviews per listings using a join between the Listing and Reviews tables. Then we order and rank the number of reviews partitioned by host\_id. Eventually we select for each host the top 3 listings.

#### SQL statement

#### select HOST\_ID, LISTING\_ID

#### from(

#### select HOST\_ID, LISTING\_ID, ROW\_NUMBER() over(partition by HOST\_ID order by counted desc) as r

#### from

#### (select distinct(l.LISTING\_ID), l.HOST\_ID, count(\*) over(partition by l.LISTING\_ID) as counted

#### from LISTING l, REVIEWS r

#### where l.LISTING\_ID = r.LISTING\_ID

#### )

#### )

#### where r <= 3;

Result:

host id listing id

2217 2015

2217 21315310

2217 18773184

3073 6287375

3718 3176

Query 7:

#### Description of logic:

First we filter the listings in Berlin that are private rooms. Then we count the amenities partitioned by neighborhood with respect with the amenity name of those listings. Eventually we sort and rank the amenities based on the count and display those with rank <=3.

#### SQL statement

#### select AMENITY\_NAME, NEIGHBORHOOD

#### from

#### (select AMENITY\_NAME, AMEN\_COUNT, NEIGHBORHOOD , row\_number() over(partition by ordered\_data.NEIGHBORHOOD order by AMEN\_COUNT desc) as rank

#### from

#### (select distinct AMENITY\_NAME, AMEN\_COUNT, NEIGHBORHOOD

#### from

#### (select AM.AMENITY\_NAME, count(AMENITY\_NAME) over(partition by LOC.NEIGHBORHOOD, AM.AMENITY\_NAME) as amen\_count, LOC.NEIGHBORHOOD

#### from LISTING\_LOCATION LOC, AMENITIES AM, LISTING\_AMENITIES LA

#### where LOC.LISTING\_ID = LA.LISTING\_ID and LA.AMENITY\_ID = AM.AMENITY\_ID and LOC.LISTING\_ID in (

#### select L.LISTING\_ID

#### from LISTING L, MATERIAL\_DESCRIPTION MD

#### where L.LISTING\_ID = MD.LISTING\_ID and L.CITY = 'Berlin' and MD.ROOM\_TYPE = 'Private room'

#### )

#### ) data\_amen

#### order by data\_amen.NEIGHBORHOOD, data\_amen.amen\_count desc) ordered\_data

#### ) ranked\_data

#### where ranked\_data.rank <= 3 ;

Result:

Essentials Adlershof

Heating Adlershof

Wifi Adlershof

Heating Alt-Hohenschönhausen

Essentials Alt-Hohenschönhausen

Wifi Alt-Hohenschönhausen

Wifi Alt-Treptow

Essentials Alt-Treptow

Query 8:

#### Description of logic:

First we created a view with the hosts and their respective verification count using count and group by host id in the host verification table.

Then we order this list by descending order (host with the most diverse way), take the first row and compute (using the review\_scores) the average communication review scores for that host.

We do the same (with an inverted ordering) for the host with the least diverse way of verification.

Eventually we compute the difference in a select statement.

#### SQL statement

#### create view number\_of\_host\_verif as

#### select count(\*) as verifications, HV.HOST\_ID

#### from HOST\_VERIFICATIONS HV

#### group by HV.HOST\_ID;

#### select avg(average\_most.avg\_m - average\_least.avg\_l) as diff

#### from

#### 

#### (select coalesce(avg(RS1.REVIEW\_SCORES\_COMMUNICATION),0) as avg\_m

#### from

#### 

#### (select h.HOST\_ID

#### from

#### (select n.HOST\_ID

#### from number\_of\_host\_verif n

#### order by n.verifications desc) h

#### where rownum = 1) host\_most ,

#### 

#### LISTING L1, REVIEWS\_SCORES RS1

#### 

#### where L1.LISTING\_ID = RS1.LISTING\_ID and L1.HOST\_ID = host\_most.HOST\_ID and RS1.REVIEW\_SCORES\_COMMUNICATION is not null

#### ) average\_most

#### ,

#### (select coalesce(avg(RS2.REVIEW\_SCORES\_COMMUNICATION),0) as avg\_l

#### from

#### 

#### (select h2.HOST\_ID

#### from

#### (select n2.HOST\_ID

#### from number\_of\_host\_verif n2

#### order by n2.verifications asc) h2

#### where rownum = 1) host\_least ,

#### 

#### LISTING L2, REVIEWS\_SCORES RS2

#### 

#### where L2.LISTING\_ID = RS2.LISTING\_ID and L2.HOST\_ID = host\_least.HOST\_ID and RS2.REVIEW\_SCORES\_COMMUNICATION is not null

#### ) average\_least;

Result:

10 (the second host has apparently no communication review and the first one has 10)

Query 9:

#### Description of logic:

First we compute what are the room types that have an average of accommodate > 3 and then we count the number of review per listings and then sum them with respect to the cities.

We then rank the cities based on that sum and select the first one.

#### SQL statement

#### select \* from (

#### select city from

#### (select sum(rev\_per\_list) over(partition by CITY) as total, CITY from

#### LISTING l,

#### (select distinct(rev.LISTING\_ID), count(\*) over(partition by rev.LISTING\_ID) as rev\_per\_list

#### from REVIEWS rev) rpl,

#### (select md.LISTING\_ID, AVG(md.ACCOMODATES) over(partition by md.ROOM\_TYPE) as average\_per\_room\_type

#### from MATERIAL\_DESCRIPTION md) av

#### where rpl.LISTING\_ID = av.LISTING\_ID and l.LISTING\_ID = av.LISTING\_ID and av.average\_per\_room\_type > 3)

#### order by total desc)

#### where rownum = 1;

Result:

Madrid

Query 10:

#### Description of logic:

First we filter the listings and select only those whose host have been registered before 2017-06-01. Then we remove those who were not occupied in 2019. Finally we compare the ratio of those who were occupied in 2019 vs all the listings per neighborhood (using group by) and display those who have a ratio >= 0.5.

#### SQL statement

#### select total\_listing.NEIGHBORHOOD

#### from

#### (select count(\*) as total, LOC2.NEIGHBORHOOD

#### from LISTING\_LOCATION LOC2, LISTING L3

#### where LOC2.LISTING\_ID = L3.LISTING\_ID and L3.CITY = 'Madrid' group by LOC2.NEIGHBORHOOD) total\_listing

#### ,

#### (select count(\*) as occupied\_listings, LOC.NEIGHBORHOOD

#### from LISTING\_LOCATION LOC, LISTING L2

#### where L2.LISTING\_ID = LOC.LISTING\_ID and L2.CITY = 'Madrid' and L2.LISTING\_ID in (

#### select distinct L1.LISTING\_ID

#### from LISTING L1, CALENDAR CAL

#### where CAL.CALENDAR\_DATE >= date '2019-01-01' and CAL.AVAILABLE = 'f' and L1.LISTING\_ID = CAL.LISTING\_ID

#### and L1.LISTING\_ID in (

#### select L.LISTING\_ID

#### from LISTING L, HOST H

#### where L.CITY = 'Madrid' and L.HOST\_ID = H.HOST\_ID and H.HOST\_SINCE <= date '2017-06-01'

#### )

#### ) group by LOC.NEIGHBORHOOD) filtered\_listing

#### where total\_listing.NEIGHBORHOOD = filtered\_listing.NEIGHBORHOOD and (filtered\_listing.occupied\_listings / total\_listing.total) >= 0.5;

Result:

Malasaña

Prosperidad

Cortes

San Blas

La Chopera

Berruguete

Bellas Vistas

Query 11:

#### Description of logic:

We first select the listings that were available in 2018 and then group them by country. Then we compute the total number of listings per country and compute the ratio between those that were available in 2018 and the total number and all of that with respect with the countries (using group by). Eventually, we filter out the countries whose ratio are below 0.2.

#### SQL statement

#### select filtered.COUNTRY

#### from

#### (select count(\*) as available, L.COUNTRY

#### from LISTING L

#### where L.LISTING\_ID in (

#### select distinct L1.LISTING\_ID

#### from LISTING L1, CALENDAR CAL

#### where CAL.CALENDAR\_DATE >= date '2018-01-01' and CAL.CALENDAR\_DATE < date '2019-01-01' and CAL.AVAILABLE = 't' and L1.LISTING\_ID = CAL.LISTING\_ID

#### 

#### ) group by L.COUNTRY) filtered

#### 

#### ,

#### (select count(\*) total\_listing, L2.COUNTRY

#### from LISTING L2

#### group by L2.COUNTRY) total

#### where filtered.COUNTRY = total.COUNTRY and (filtered.available/ total.total\_listing) >= 0.2;

Result:

Spain

Germany

Query 12:

#### Description of logic:

First we filter the listings that are strict with grace periods per neighborhood and then compute the ratio per neighborhood and filter out if the ratio is smaller than 0.5.

#### SQL statement

#### select total.NEIGHBORHOOD

#### from

#### (select count(\*) total\_list, LOC.NEIGHBORHOOD

#### from LISTING\_LOCATION LOC, LISTING L

#### where LOC.LISTING\_ID = L.LISTING\_ID and L.CITY = 'Barcelona' group by LOC.NEIGHBORHOOD) total

#### ,

#### (

#### select count(\*) strict\_count, LOC.NEIGHBORHOOD

#### from LISTING\_LOCATION LOC, LISTING\_DETAILS LD, LISTING L2

#### where LOC.LISTING\_ID = LD.LISTING\_ID and L2.LISTING\_ID = LD.LISTING\_ID and L2.CITY = 'Barcelona' and LD.CANCELLATION\_POLICY = 'strict\_14\_with\_grace\_period' group by LOC.NEIGHBORHOOD

#### ) filtered

#### where total.NEIGHBORHOOD = filtered.NEIGHBORHOOD and (filtered.strict\_count / total.total\_list) >= 0.05;

Result:

Glòries - El Parc

La Nova Esquerra de l'Eixample

L'Antiga Esquerra de l'Eixample

Sant Pere/Santa Caterina

Sarrià

Sant Gervasi - la Bonanova

Query Analysis

Selected Queries (and why)

We selected query 12, 3 and 2 because we could easily improve them by putting indexes on the table’s access predicates.

The optimized queries toghether with other relevant informations can be found here :

<https://github.com/hedi-sassi/rbnb_db_project/blob/master/optimized_queries.sql>

#### Query 12

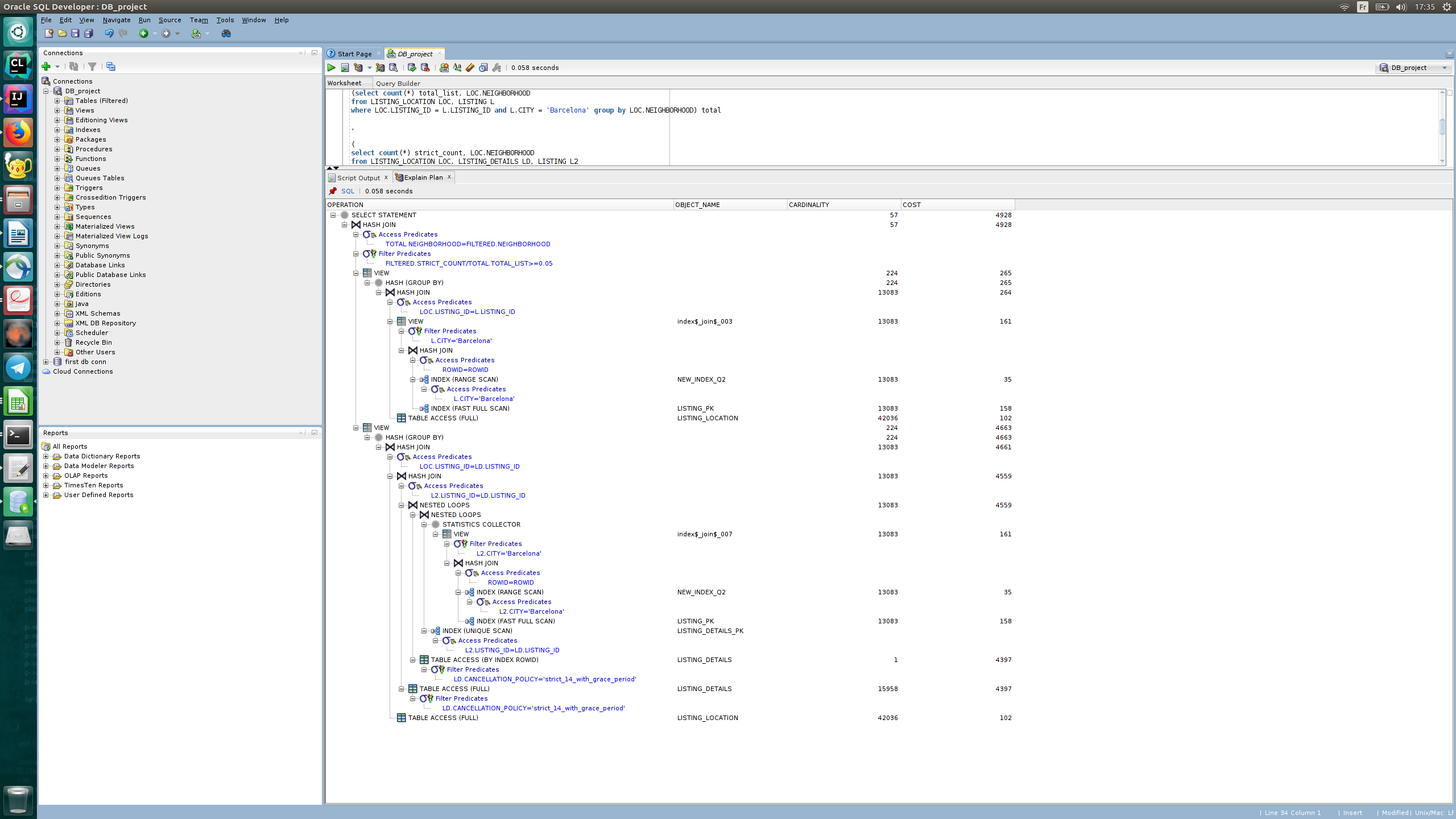
Initial Running time: 0.12 s

Optimized Running time: 0.064 s

Explain the improvement:

We put indexes on the listing city and the cancellation policy. Thus table access predicates are faster since there is an index on the filtered columns.

Initial plan:



Improved plan:

#### Query 3

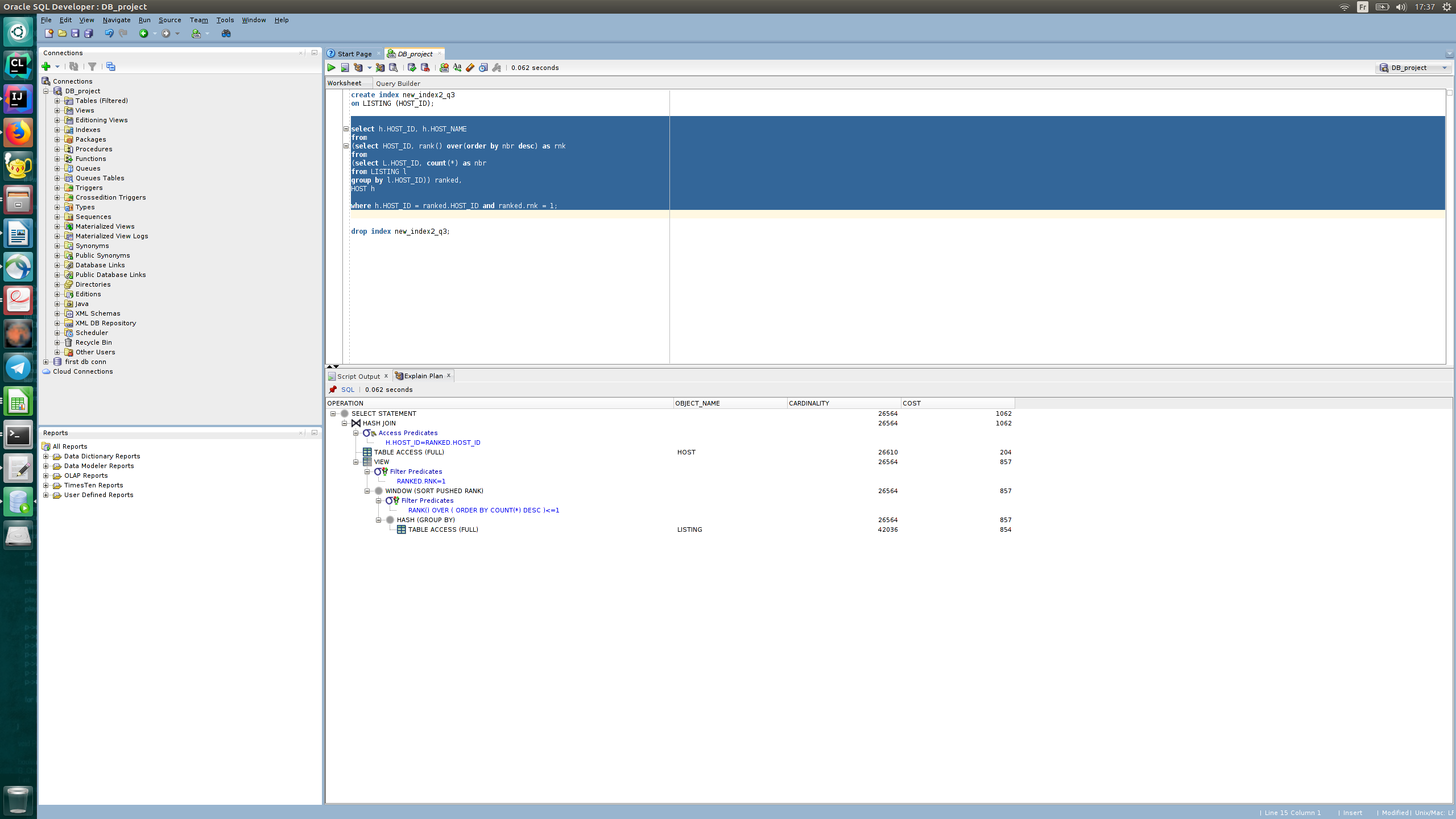
Initial Running time: 0.044 s

Optimized Running time: 0.035 s

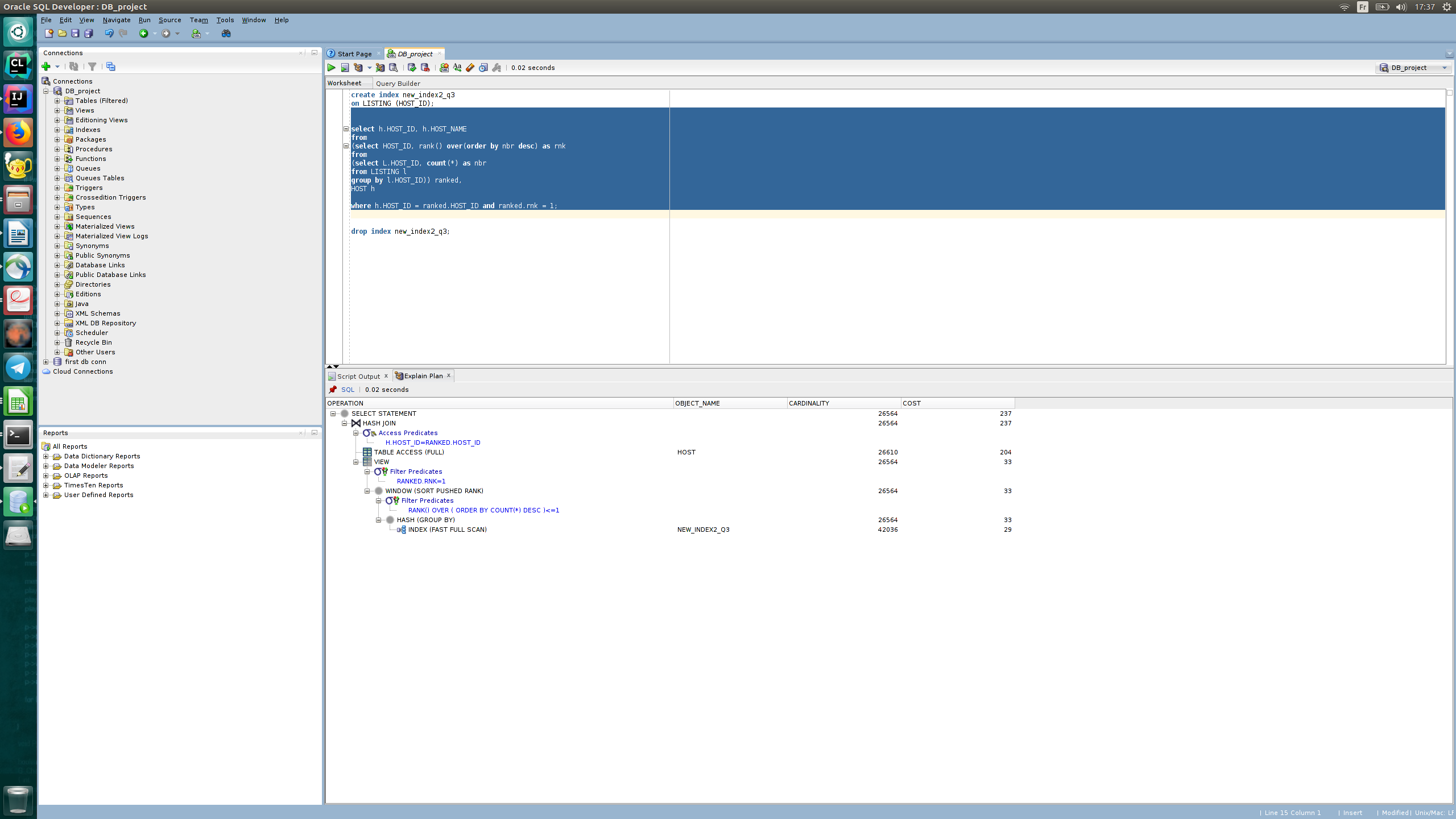
Explain the improvement: the join predicate is h.host\_id = ranked(listing).host \_id but since listings don't have an index on the host id => lose performance.

We only have to add an index on listing.host\_id.

Initial plan:



Improved plan:



#### Query 2

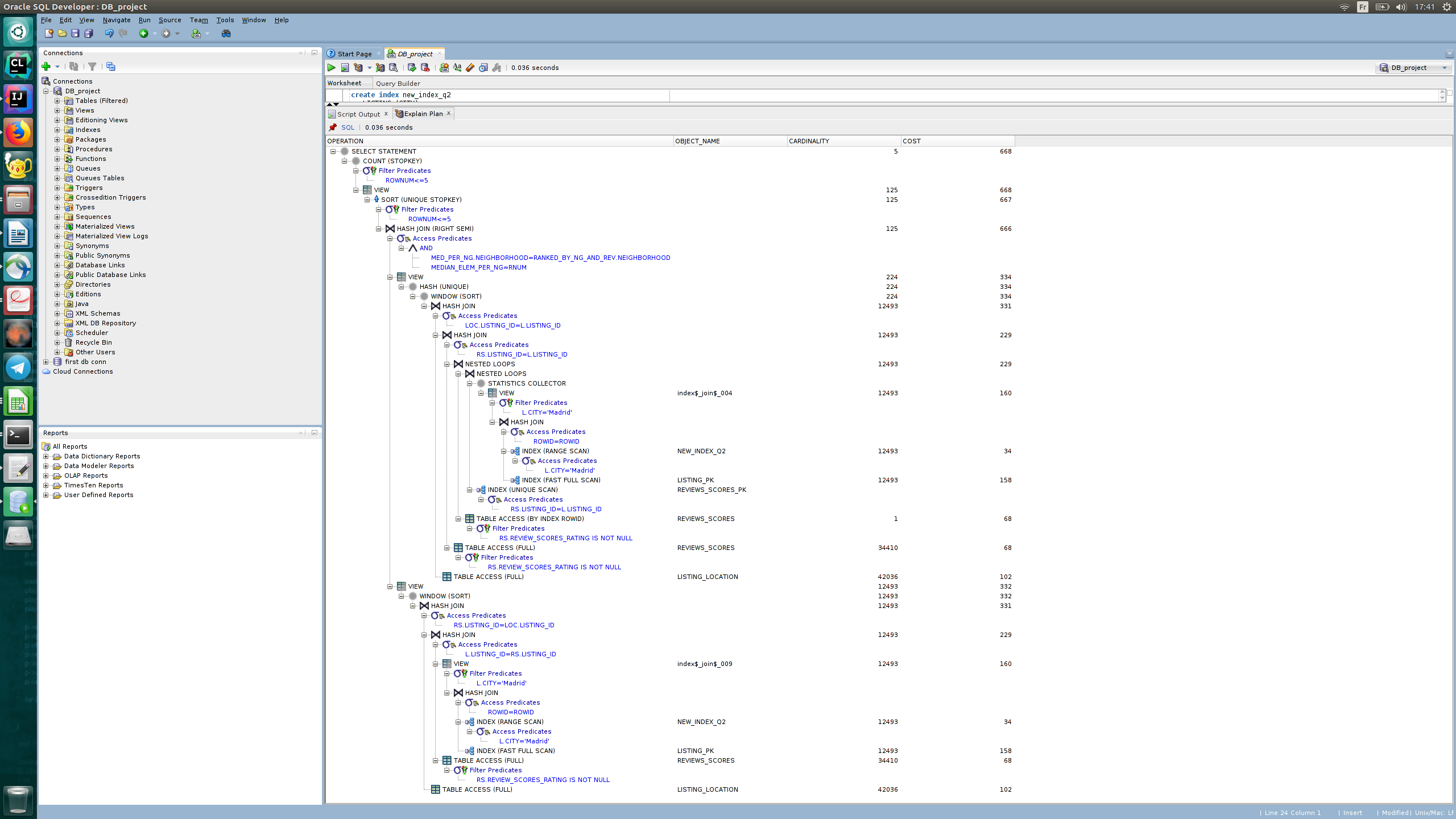
Initial Running time: 0.08 s

Optimized Running time: 0.065 s

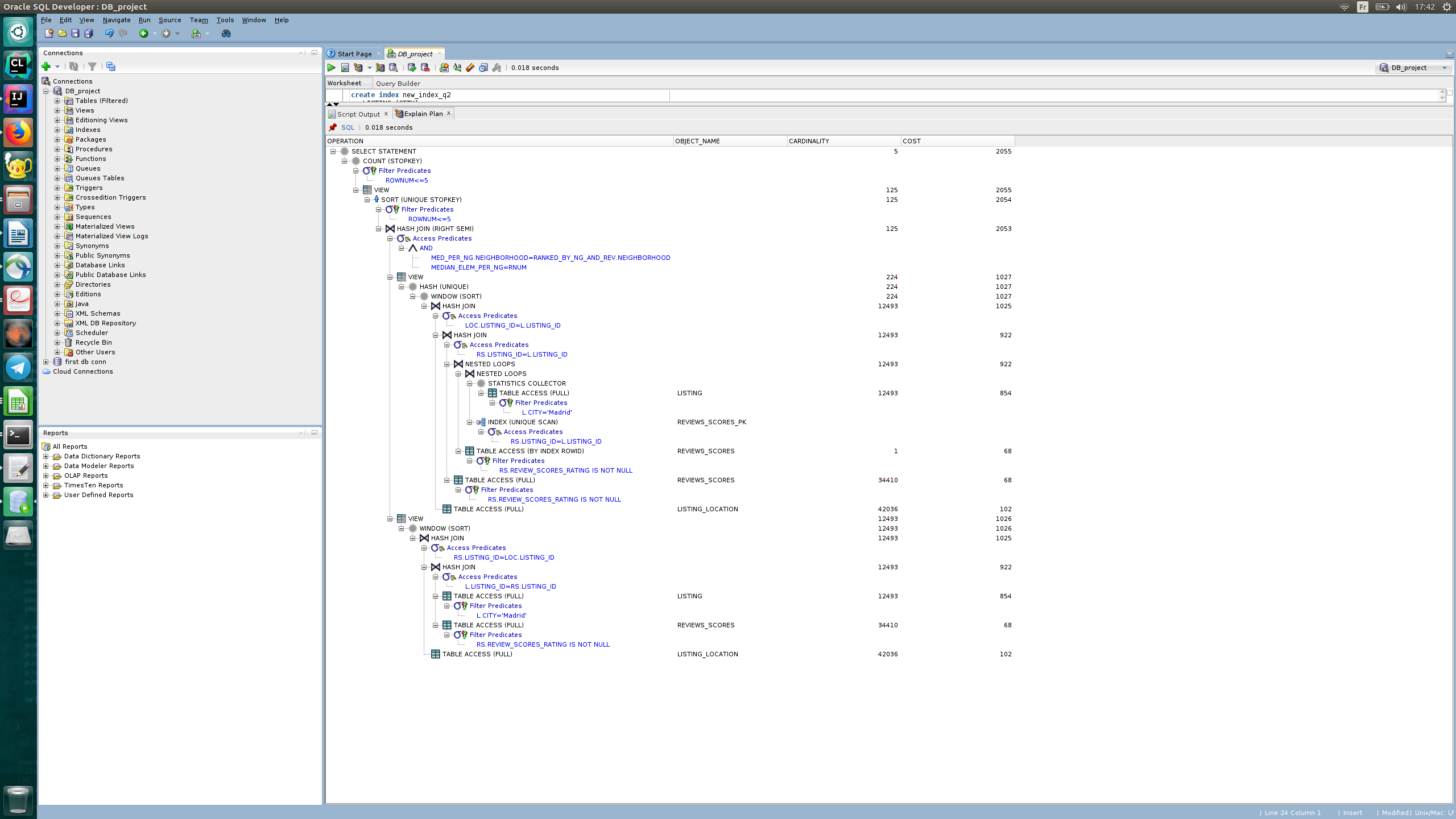
Explain the improvement:

Putting an index on the city facilitates the access to the listing table.

Initial plan:



Improved plan:



Interface

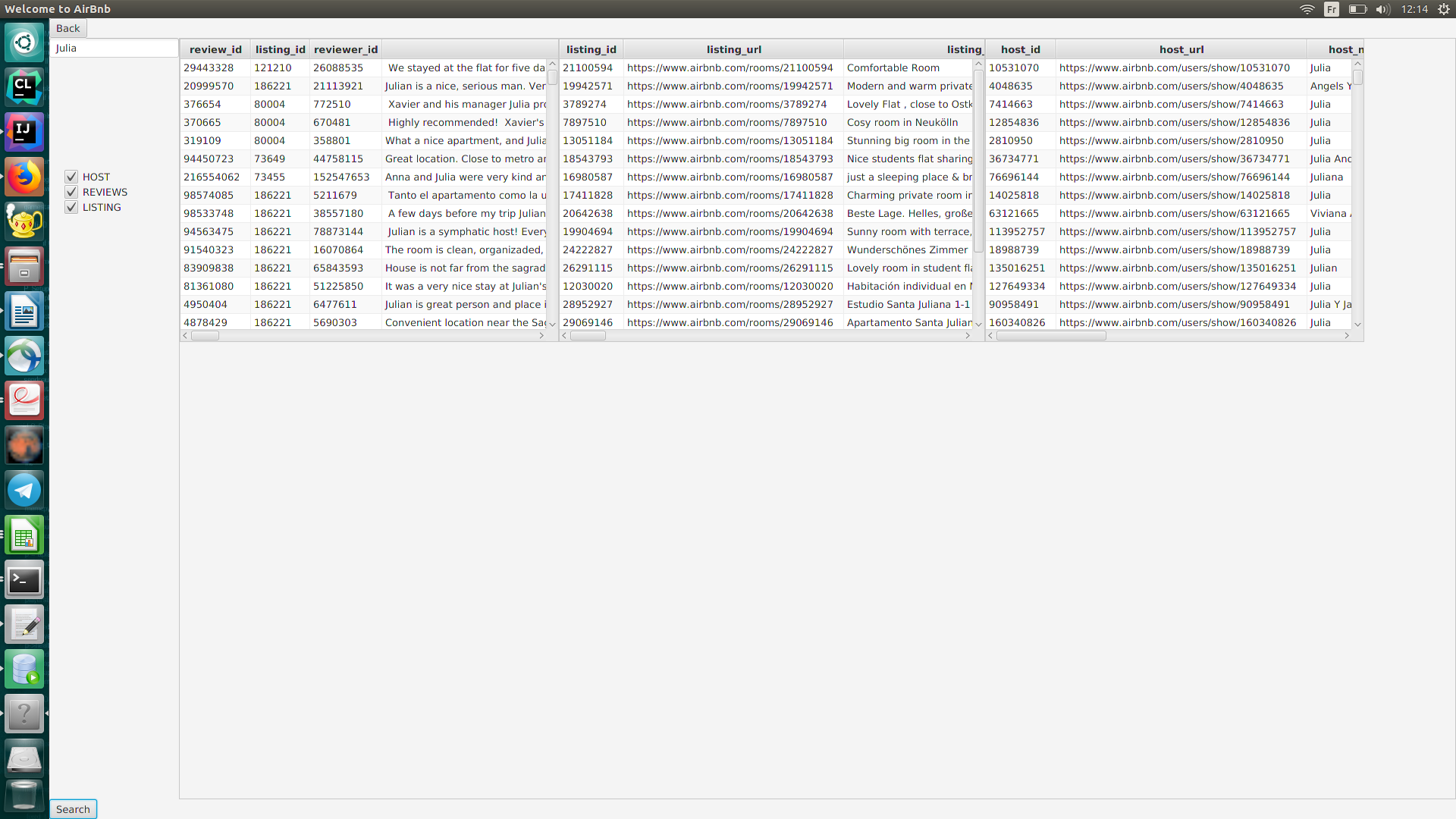
Design logic Description

This part is similar to the one in milestone 2.

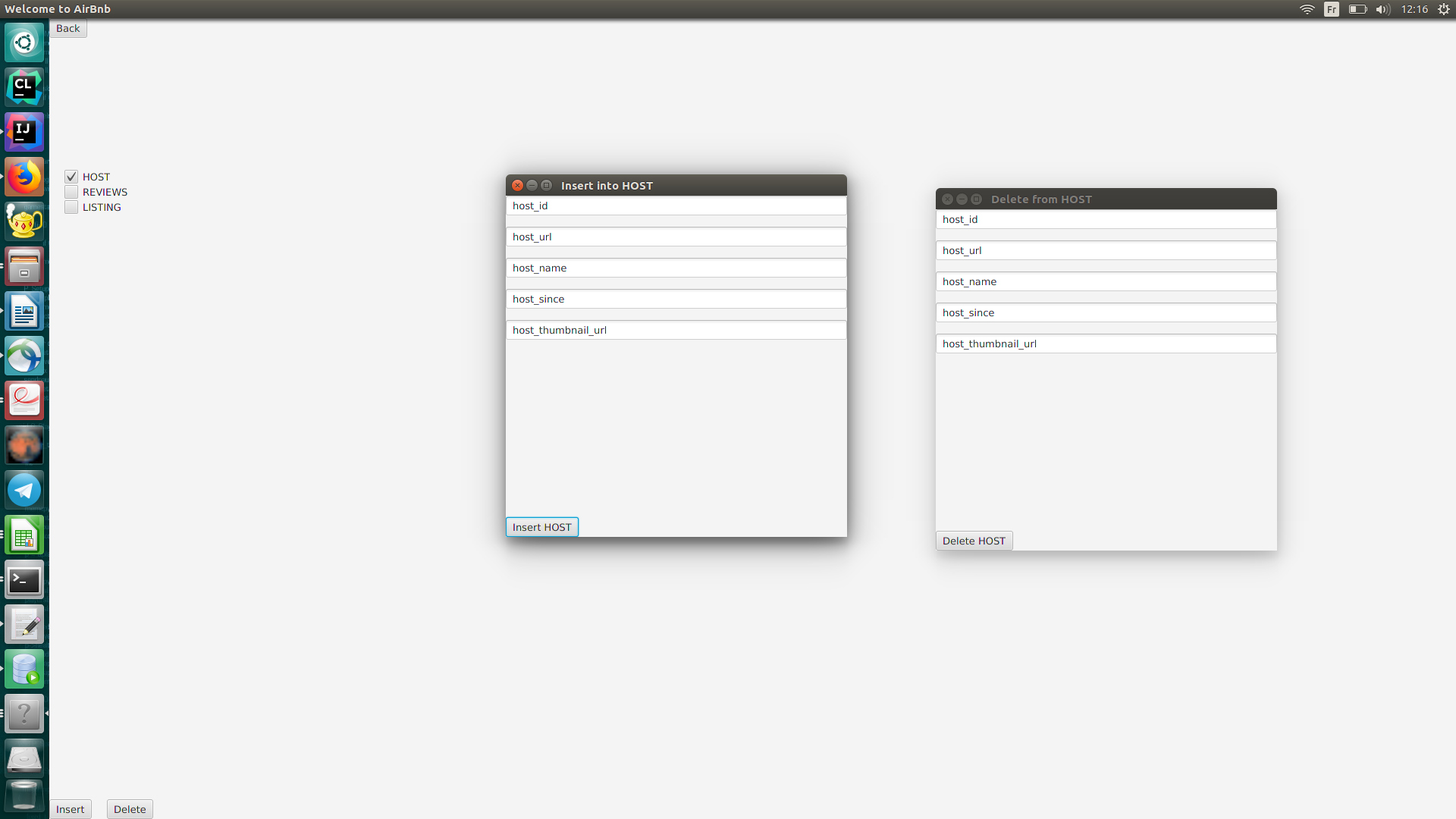
We used JavaFx and JDBC. The design is simple: a main page with buttons to switch to action-specific windows (insert/delete, search and predefined queries).

The code can be found here: <https://github.com/hedi-sassi/rbnb_db_project/tree/master/UI>

Screenshots



Here we see the results of a search in 3 different tables.



This is the insertion/deletion window.

General Comments

We split the work as follows:

Simon, Hédi & Camilla : writing queries and queries optimization

Hédi : Report and UI

Camilla: Correction of queries from milestone 2